On Soft Predicates in Subdivision Motion Planning*

[Extended Abstract]

Cong Wang
Dept. of Computer Science and Engineering
Polytechnic Institute of NYU
Brooklyn, NY, USA
cwang05@students.poly.edu

Yi-Jen Chiang
Dept. of Computer Science and Engineering
Polytechnic Institute of NYU
Brooklyn, NY, USA
yjc@poly.edu

Chee Yap
Dept. of Computer Science
New York University
New York, NY, USA
yap@cs.nyu.edu

ABSTRACT

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 $SE(2) = \mathbb{R}^2 \times S^1$. To validate this approach, we implement our algorithms and the experiments demonstrate the efficiency of our approach, even compared to probabilistic algorithms.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations—*computational geometry.*

General Terms

Algorithms, Theory, Experimentation.

*This work is supported by NSF Grant CCF-0917093 and DOE Grant DE-SC0004874.

SoCG'13, June 17 - 20, 2013, Rio de Janeiro, Brazil.

Keywords

computational geometry; exact algorithms; subdivision algorithms; motion planning; robotics; soft predicates; resolution-exact algorithms.

1 Introduction

A central problem of robotics is motion planning [4, 19, 20, 9]. In the early 80's there was strong interest in this problem among computational geometers [14, 30]. 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 [29] showed that the method of algebraic cell decomposition is a universal solution for motion planning. We introduced the retraction method in [23, 31, 32]. In the first survey of algorithmic motion planning [37], we also showed the universality of the retraction method. This method is now commonly known as the road map approach, popularized by Canny [7] 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 90's, interest shifted back to more practical techniques. Today, the dominant approach is based on randomization or sampling. Perhaps its most well-known representative is the **probabilistic roadmap method** (PRM) [18]. The idea is to compute a partial road map by random sampling of the C-space. PRM offers a computational framework for a class of algorithms. Moreover, many variants of the basic framework have been developed. A partial list includes Expansive-Spaces Tree planner (EST), Rapidly-exploring Random Tree planner (RRT), Sampling-Based Roadmap of Trees planner (SRT); see [20, 9]. Most sampling approaches take sample points in configuration space, but the recent paper from Halperin's group [27] takes sample (parametrized) subsets of configuration space. In an invited talk at the recent IROS 2011 Workshop on Progress and Open Problems in Motion Planning[1], 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

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

issues such as the "Climber's Dilemma" [15, 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" [9, 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 [17] about the environment. It is unclear how to check these assumptions on specific environments. For a comprehensive overview of motion planning, see Lavalle [20] and Choset et al. [9].

In this paper, we turn to a third popular approach [43] 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 bisection search ($d = 1$) or quad-trees ($d = 2$). An early reference for this approach is Brooks and Lozano-Perez [5]. Recent subdivision references include [43, 2, 42, 11, 25]. Manocha's group has been active and highly successful in producing practical subdivision algorithms for a variety of tasks, many critical in motion planning [35, 34]. Domain subdivisions are sometimes known as "cell decomposition" (e.g., [43]), but we reserve this term for the algebraic approaches based on partitioning the configuration space into algebraic cells that are directly correlated with the combinatorial features on the obstacles (e.g., [28, 37]). 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 [42, 43]. Such kinds of subdivision algorithms offer tantalizing opportunities for new kinds of complexity analysis. Examples of such analysis may be seen in [26, 33, 6].

¶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 [9] but rarely analyzed (Section 5 discusses why). Our first contribution is to introduce the related concept of **resolution-exact** (or **ε-exact**) **planners**. Such planners accept an input **resolution parameter** $\varepsilon > 0$. There is an **accuracy constant** $K > 1$ such that if there is a path of clearance $K\varepsilon$, it will output a path; if there is no path of clearance ε/K, it will output "NO PATH". Thus we avoid the halting problem of probabilistically complete planners like PRM. As noted in Section 5, it is not automatic that "resolution completeness" solves the halting problem. Furthermore, our definition is not a "trick" to avoid the halting problem by fiat. When we output "NO PATH", it guarantees that there are no paths of clearance $K\varepsilon$; no such information can come from PRM. Our ε parameter has practical significance: good engineers know the limits of accuracy of their sensors, controls, etc. Path planning that depends on accuracy beyond these limits is not realistic, even dangerous. We can choose ε based on such engineering limits such that, when we declare "NO PATH", no further search is warranted. There are subtleties and interesting variations in the concept of resolution-exactness (Section 5). For instance, we prove that there is inherent indeterminacy in such algorithms.

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. Although such primitives are perhaps nascent in previous literature, by making this idea explicit, we open up new possibilities, as well as lay the groundwork for a systematic investigation of 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 $SE(2)$. Our experiments demonstrate their effectiveness.

Due to space limitation, proofs are deferred to the full paper.

2 On Numerical Subdivision Algorithms

Computational Geometry 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 of Exact Methods to have wider impact on robotics and fields of Computational Sciences and Engineering (CS&E) where geometric reasoning is dominant calls for a re-examination of our assumptions. We argue that Subdivision Algorithms, when[2] combined with soft primitives, offer a pathway for Computational Geometers to design new algorithms that are theoretically sound and practical. Our soft primitives do not entail any error analysis in the style of numerical analysis; rather, we rely on various interval methods [21]. For a general discussion, see [40].

One limitation of numerical primitives is that they are only complete in the limit. They also cannot detect degeneracies unless we use zero bounds [39]. But these are not an issue for resolution-exact planners. On the other hand, numerical methods are more general, applicable to analytic (non-algebraic) problems where exact solutions are generally unknown [8].

The current limitations of the Subdivision Methods are that while practical Sampling Methods have been applied to problems with high (say dozens) degrees of freedom (DOF), this has not been done with Subdivision Methods. Hence the conventional wisdom of roboticists is that Subdivision Methods are effective only up to medium degrees of freedom. We believe that this conventional wisdom can be overcome with better (or even randomized) subdivision strategies. Note that the size of subdivision trees is not necessarily exponential in the depth or resolution if we subdivide adaptively; in 1- and 2-dimensions such subdivision trees for root isolation are provably optimal up to logarithmic terms [26, 33].

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 $r_0 > 0$.

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

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 [23]. For polygonal Ω, the roadmap is efficiently computed as the Voronoi diagram of line segments [38, 12]. 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 [16, 36]. 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** (such box contains at least one semi-free configuration, but possibly free and stuck configurations as well). 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 $Sep(A, B) := \inf\{\|a - b\| : a \in A, b \in B\}$. 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.

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** $ROI(\mathcal{T})$ of the tree. Subsequently, $\mathcal{T}$ can be **expanded** at any `ON`-leaf B, by **splitting** B into 2^i $(1 \leq i \leq d)$ congruent subboxes who become the children of B.

¶3. An Exact Subdivision 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 $\texttt{S}plit(B, \varepsilon)$ which returns a subdivision of B into 2^i (for some $i = 0, \ldots, d$) congruent subboxes; the split subroutine is said to *fail* if $w(B) \leq \varepsilon$ (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 $\mathcal{T}$. We define an undirected graph $G(\mathcal{T})$ with 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 $\texttt{S}plit(B, \varepsilon)$. If $\texttt{S}plit(B, \varepsilon)$ 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 $\texttt{S}plit(B, \varepsilon)$ is made a child of B. We label B' with the predicate $C(B')$. If $C(B') = \texttt{FREE}$, 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 $C(B') = \texttt{MIXED}$ and $w(B') > \varepsilon$, we insert B' into Q. Thus, mixed boxes of width $\leq \varepsilon$ are discarded (effectively regarded as `STUCK`). Now we are ready to present a simple but useful exact subdivision algorithm:

Exact FindPath:
Input: Configurations α, β, tolerance $\varepsilon > 0$, box $B_0 \in \mathbb{R}^d$.
Output: Path from α to β in $Free(R_0, \Omega) \cap B_0$.
Initialize a subdivision tree $\mathcal{T}$ with only a root B_0.
1. While $(Box_{\mathcal{T}}(\alpha) \neq \texttt{FREE})$
   If (Expand $Box_{\mathcal{T}}(\alpha)$ fails) Return("No Path").
2. While $(Box_{\mathcal{T}}(\beta) \neq \texttt{FREE})$
   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 $(B_1, \ldots, B_m)$ of free boxes where B_i, B_{i+1} are adjacent. We convert the 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. Note that our channels are free, in contrast to the M-channels (each a sequence of adjacent `FREE` or `MIXED` leaf boxes) of Zhu-Latombe [43], Barbehenn-Hutchinson [2] and Zhang-Manocha-Kim [42]. Freeness is essential for Union-Find.

The routine $\mathcal{T}.getNext()$ in Step (*) is not fully specified, but critical. To ensure "completeness" of this algorithm, a simple solution is to return any mixed leaf of minimum

depth. Below, we will provide careful analysis of completeness. But many other interesting heuristics are possible: If $getNext()$ is random, we obtain a form of Sampling Method. By alternating between randomness and some deterministic strategy, we can get the best of both worlds. If $getNext()$ always return a mixed leaf that is adjacent to the connected component of $Box_{\mathcal{T}}(\alpha)$, we get a sort of Dijkstra's algorithm or A*-search (see Barbehenn and Hutchinson [2, 1]). Another idea is to use some entropy criteria. Recent work on shortest-path algorithms in GIS road systems offers many other heuristics. The use of Union-Find is natural and proposed in [20].

The above exact subdivision is not our claim to novelty. Nevertheless, our framework has interesting features, and offers potentially great adaptivity through its $getNext()$ strategy. For instance, the (uniform) grid [20, p. 185] is widely used. Although grids are superficially similar to subdivisions, grids use point-based operations while our theory is based on box (interval) operations (see Sec. 4). Uniform grid translates into breadth-first search strategy for $getNext()$, but we can do much better. Zhu and Latombe [43] propose a goal-directed form of $getNext()$: pick some "shortest" M-channel (sequence of adjacent `FREE` or `MIXED` leaf boxes) and expand all the `MIXED` boxes in the channel. Barbehenn and Hutchinson continued this strategy but introduced the highly efficient Dijkstra-search or its extension to A* search [2, 1]. While finding the shortest A*-path is efficient, the efficient update of the A*-structure after expansion is not well-understood.

4 Let us Design Soft Predicates!

¶4. Soft Predicates. In our subdivision framework, our true interest lies in replacing the exact predicate $C(B)$ by some soft version $\widetilde{C}(B)$ which is easy to compute and "correct in the limit". We formalize the needed properties. 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 **safe**, 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 and $C(\gamma) \neq \texttt{MIXED}$, then $\widetilde{C}(B_i) = C(\gamma)$ for large enough i.

We need a quantitative measure of the convergence rate. Let $0 \leq \sigma \leq 1$ 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 σ-effective for the class $\mathcal{B}$ of square boxes.

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 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

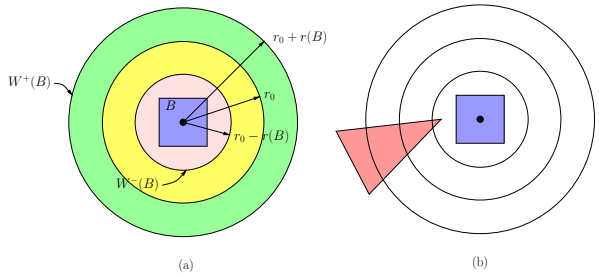

Figure 1: (a) Domains $W^+(B)$ and $W^-(B)$. (b) Condition (S1) holds.

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 Figure 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 the Figure 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. `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 $\phi(B.parent)$, we can find the feature f in $\phi(B.parent)$ that is closest to $m(B)$. We have 2 possibilities: (1) f is an edge. Assume that edges are locally oriented so that we can decide using a standard 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 $\phi(B.parent)$, and for *each* obstacle polygon S appearing in $\phi(B.parent)$, 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.

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 $\sigma = 1/\sqrt{2}$.*

All we do is to substitute $\widetilde{C}$ for C in the exact algorithm of the previous section to get a complete motion planning algorithm — this will be proved below, when we introduce resolution-exact algorithms.

¶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

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

where $Sep(A, B)$ 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` [13], `MPFR` [22]). 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:

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

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 $C(B)$.*

¶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 [36]). 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. They are "resolution complete" in the informal sense of the literature. But what exactly does this mean? A common definition in the literature says that *"Resolution completeness" is the property that the planner is guaranteed to find a path if the resolution of an underlying grid is fine enough.* Two questions arise: what is "fine enough" and what happens if there is no path? Presumably, "fine enough" means "as the resolution parameter h goes to 0". But if h is not bounded away from 0, this entails an infinite search and such algorithms will suffer from the halting problem that plaques Sampling Methods.

Notice that 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 another attempt to define resolution completeness, where we now state a converse condition for "no path': (i) *if there is a path with clearance ε, then the algorithm 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 correct, as it implies that we can detect the case where the clearance is exactly ε, a feat that only Exact Methods can perform (in which case we might as well design algorithms with $\varepsilon = 0$). What is missing in current discussions of resolution completeness is an **accuracy parameter** $K > 1$. We say that a planner has an **accuracy** $K > 1$ if the following holds:

- 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".

Now we can define a concept noted in the introduction: a planner is said to be **resolution-exact** if it has an accuracy $K > 1$. 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 before in ¶1, there is no infinite regress if we know some hard engineering limits of how much clearance a path should have. The indeterminacy depends on the accuracy parameter K of our algorithm.

The result of Theorem A below concerns our algorithm Exact FindPath in ¶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 $\varepsilon/2 < t \leq \varepsilon$. Now consider the "full expansion" of the subdivision tree $\mathcal{T}$ whose leaves are of the smallest size possible. Recall from ¶3 that a channel is a sequence $(B_1, \ldots, B_m)$ 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 $\varepsilon/4$.*

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 Figure 2). 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

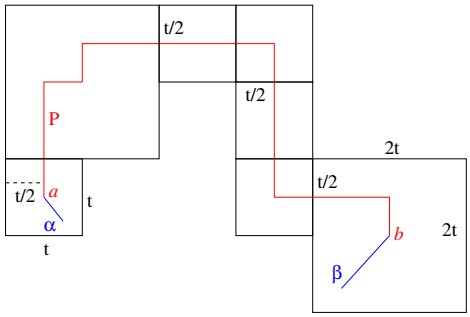

Figure 2: Canonical path P^* contains the essential path P (red) with essential clearance $t/2$.

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.

LEMMA 4. *If there is no free canonical path with essential clearance $\varepsilon/4$, then our* EXACT FINDPATH *algorithm reports "no path".*

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 $K_0, k_0 \geq 1$ 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.

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. Third is the use of soft predicates. In particular, what is the accuracy of our prototype algorithm in ¶3 when using the soft predicates of ¶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 $\sigma\varepsilon/4$.*

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".*

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.

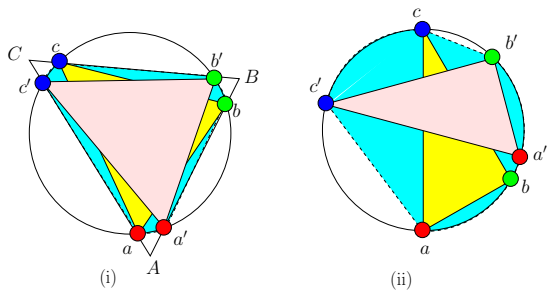

Figure 3: Shaded areas represent round triangles: (i) $aa'bb'cc'$, (ii) $ab'cc'$. In (i), the round triangle $aa'bb'cc'$ is $T \cap D$ where T is the triangle (A, B, C) and D is the (white) disk.

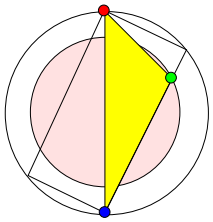

Figure 4: Enclosing circle of enclosing rectangle for obtuse triangle: their rotation.

¶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 $w(R) \geq r_0$, 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 $w(R) \leq \varepsilon$. Let us say that R is ε-**small** if $w(R) \leq \varepsilon$. We need a similar criterion for Θ: say Θ is ε-**small** if $|\Theta| \leq \varepsilon/r_0$. 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 $\mathtt{S}split(B, \varepsilon)$ 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 $0 < \varepsilon \leq \pi/2$. 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 $(1 + \sqrt{2})\varepsilon$.*

¶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 intersects 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. 3).

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 $s \geq 0$; 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. 3, 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. 3(i), $W^-(B)$ is the triangle bounded by the lines $\overline{ab'}$, $\overline{bc'}$ and $\overline{ca'}$.

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`[3] and will be available on our project web page. The platform for the experiments was a Linux Fedora 16 OS with a 3.4GHz Intel Quad Core CPU, and 16GB RAM. Our current implementation does not apply the technique of "lax comparison" in ¶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.

The input obstacle sets are bugtrap, input150, input200, and input300, each represented by a set of polygons (not necessarily disjoint), with the dimension of the global environment 512 x 512. For "input x" we generated x triangles at random. Some images are shown in the Appendix. For each input, we show in the left table the statistics of running our planner for a given robot. The disc robot is specified as disc(r, T) where r is the robot radius and $T \in \{B, G, R\}$ indicates the search strategy (B = *Breadth First Search* (BFS), G = *Greedy Best First* (GBF), R = *Random*). Similarly, the triangle robot is specified by tri(r, T). In general, we found GBF to be the fastest. Whenever the randomized strategy $T = R$ is used, the statistics is the average of 5 runs; these are reproducibly encoded in Makefile targets. We have columns reporting the number of free, stuck, and mixed boxes. There were two kinds of mixed boxes: those of size $> \varepsilon$ and the rest. Note that when the number of mixed boxes of size $> \varepsilon$ is zero (last column), this implies 'NO PATH'. The converse is true only for the BFS or Random search strategies. Hence we explicitly mark the entries in the last column with an asterisk (*) to indicate 'NO PATH'.

We also directly compared our triangle with GBF strategy (the instances of the left-table entries in bold, also shown in Figs. 7-9 in the Appendix) with PRM, for which we ran the benchmark package OOPSMP [24]; we show the results in the right table (top), where the PRM times are shown as preprocessing, query, and total times. In general, our running times are competitive with PRM (but can be slower than PRM; see entries marked with "*" (at different α, β)), and can be much faster in some instances as shown here. OOPSMP requires user-chosen parameters like number of sample points, budgeted times for preprocessing and query (we used the default values: 5000 points, 5s and 5s). Our only parameter is $\varepsilon > 0$. Finally, we compared our disc robot (disc(15, G), $\varepsilon = 0.5$) on bugtrap with PRM (robot must be a polygon, approximated by a same-radius regular 20-gon; default settings except for 25000 samples as no path was found for 5000 samples). We show the results in the right table (bottom); clearly we are significantly faster.

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 [17] are non-verifiable. This paper takes up the classic subdivision paradigm to develop

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

Obstacle (input)	robot (radius)	eps	time (ms)	free	stuck	mixed $< \varepsilon$	mixed $\geq \varepsilon$
bugtrap	disc(14,G)	1	16	3867	2076	3403	462
	disc(14,G)	2	10	1779	943	1750	275
	disc(14,G)	4	5	854	460	801	151 (*)
	disc(40,B)	1	24	6302	6499	6826	0 (*)
	tri(40,G)	17	**116**	14969	0	40234	6000 (*)
	tri(14,G)	4	322	64761	0	0	117517
input150	disc(7,R)	5	3	1	2	2	17 (*)
	disc(7,R)	2	428	6892	10082	8027	1955
	tri(7,G)	5	**10**	945	0	1334	360
	tri(7,B)	5	1349	152841	366	0	608432
	tri(7,R)	5	315	32179	1028	101322	32477
input200	disc(5,B)	2	16	2590	4891	0	5636
	tri(5,G)	2	**89**	16866	160	0	29602
	tri(5,B)	2	1742	182866	1036	0	747445
	tri(5,R)	2	3940	331830	7044	0	1408722
input300	disc(7,B)	4	23	3785	11284	7465	0 (*)
	disc(7,B)	1	35	6439	15339	0	11052
	tri(7,G)	4	**32**	5212	0	0	11686
	tri(7,B)	4	2101	110005	667	0	899054
	tri(7,R)	4	7470	371119	8539	0	2694907

The effect of increasing ε is seen in the first four lines in the left table. In the right table, inputs (i), (ii) differ in the α, β positions; the run-time of the winner is shown in bold.

Obstacle input file (robot radius)	Tri(GBF) (ms)	PRM Prop. (ms)	PRM Query (ms)	PRM Total (ms)
bugtrap (40)	**116**	161	4	165
input150 (7)(i)	**10**	176	2	178
input150 (7)(ii)	730 *	176	2	**178**
input200 (5)	**89**	203	5	208
input300 (7)(i)	**32**	145	0.3	145.3
input300 (7)(ii)	478 *	145	0.3	**145.3**

Obstacle input file (robot radius)	Disc(GBF) (ms)	PRM Prop. (ms)	PRM Query (ms)	PRM Total (ms)
bugtrap (15)	**30**	2135	9	2144

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. [42], 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 [42]). 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 [41] where many of the ideas here are generalized. (2) We can extend our work to subdivision of $SE(3) = \mathbb{R}^3 \times S^3$, and believe this too can be competitive with PRM. Note that no general exact algorithms have been implemented for $SE(3)$. (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 [10]. (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.

9 References

[1] M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE Trans. Robotics and Automation*, 11(2), 1995.

[2] M. Barbehenn and S. Hutchinson. Toward an exact incremental geometric robot motion planner. In *Proc. Intelligent Robots and Systems 95.*, vol. 3, pp. 39–44, 1995.

[3] R. Bohlin and L. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In *Handbook on Randomized Computing*, pp. 221–249. Kluwer Academic Pub., 2001.

[4] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Perez, and M. Mason. *Robot Motion: Planning and Control*. MIT Press, 1982.

[5] R. A. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. 8th IJCAI Vol. 2*, pp. 799–806, San Francisco, 1983. Morgan Kaufmann Pub. Inc.

[6] M. Burr, F. Krahmer, and C. Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. *Electronic Colloquium on Computational Complexity (ECCC)*, TR09(136), December 2009.

[7] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993.

[8] E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. Yap. Shortest paths for disc obstacles is computable. In *21st SoCG.*, pp. 116–125, 2005.

[9] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

[10] C. Delfinado and H.Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geom. Design*, 12:771–784, 1995.

[11] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14:443–479, 1995.

[12] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[13] GNU MP Homepage, Since 1991. GNU MP (=GMP) is a free library for arbitrary precision arithmetic. URL `http://gmplib.org`.

[14] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 755–778. CRC Press LLC, 1997.

[15] K. Hauser. *Motion planning for legged and humanoid robots*. PhD thesis, Stanford University, Dec 2008.

[16] M. Hemmer, O. Setter, and D. Halperin. Constructing the exact Voronoi diagram of arbitrary lines in three-dimensional space. In *Algorithms – ESA 2010*, vol. 6346 of *LNCS*, pp. 398–409. Springer 2010.

[17] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int'l. J. Robotics Res.*, 25(7):627–643, 2006.

[18] L. Kavraki, P. Švestka, C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

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

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

[21] R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[22] MPFR Homepage, Since 2000. URL `http://www.mpfr.org/`. MPFR is a C++-library for multi-precision floating-point computation with exact rounding modes.

[23] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1985. Also, Chapter 6 in *Planning, Geometry, and Complexity*, eds. Schwartz, Sharir and Hopcroft, Ablex Pub. Corp., Norwood, NJ. 1987.

[24] E. Plaku, K. Bekris, and L. Kavraki. OOPS for motion planning: An online open-source programming system. In *IEEE ICRA*, pp. 3711–3716, 2007.

[25] J. H. Reif and H. Wang. Nonuniform discretization for kinodynamic motion planning and its applications. *SIAM J. Computing*, 30:161–190, 2000.

[26] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In *36th ISSAC*, pp. 353–360, 2011.

[27] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin. Motion planning via manifold samples. In *Proc. European Symp. Algorithms (ESA)*, 2011.

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

[29] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Appl. Math.*, 4:298–351, 1983.

[30] J. T. Schwartz, M. Sharir, and J. Hopcroft, editors. *Planning, Geometry and Complexity of Robot Motion*. Ablex Series in Artificial Intelligence. Ablex Publishing Corp., Norwood, New Jersey, 1987.

[31] M. Sharir, C. O'D'únlaing, and C. Yap. Generalized Voronoi diagrams for moving a ladder I: topological analysis. *Communications in Pure and Applied Math.*, XXXIX:423–483, 1986.

[32] M. Sharir, C. O'D'únlaing, and C. Yap. Generalized Voronoi diagrams for moving a ladder II: efficient computation of the diagram. *Algorithmica*, 2:27–59, 1987.

[33] V. Sharma and C. Yap. Near optimal tree size bounds on a simple real root isolation algorithm. In *37th ISSAC*, pp. 319 – 326, 2012.

[34] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Proc. Symp. on Geometry Processing (SGP'04)*, pages 235–244, 2004.

[35] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graph. Models*, 68(4):343–355, 2006.

[36] C. Yap, V. Sharma, and J.-M. Lien. Towards Exact Numerical Voronoi diagrams. In *9th Proc. Int'l. Symp. of Voronoi Diagrams in Science and Engineering (ISVD)*, pp. 2–16. IEEE, 2012.

[37] C. K. Yap. Algorithmic motion planning. In J. Schwartz and C. Yap, editors, *Advances in Robotics, Vol. 1: Algorithmic and geometric issues*, volume 1, pages 95–143. Lawrence Erlbaum Associates, 1987.

[38] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram for a set of simple curve segments. *Discrete and Comp. Geom.*, 2:365–394, 1987.

[39] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.

[40] C. K. Yap. In praise of numerical computation. In äher, eds., *Efficient Algorithms*, vol. 5760 of *LNCS*, pp. 308–407. Springer, 2009.

[41] C. K. Yap. Theory of Soft Subdivision Search and Motion Planning, 2012. Submitted, April 20, 2012: Symp. on Geometric Processing (SGP).

[42] L. Zhang, Y. J. Kim, and D. Manocha. Efficient cell labelling and path non-existence computation using C-obstacle query. *Int'l. J. Robotics Research*, 27(11–12), 2008.

[43] D. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7:9–20, 1991.

APPENDIX: Images from Experiments

Fig. 5 shows the output from our algorithm for a disc robot. The leaves of the subdivision tree form a subdivision of the root box B_0. Since $C_{space} = \mathbb{R}^2$, it is easy to visualize this subdivision: each leaf box is classified as FREE/STUCK/MIXED, and color coded as indicated. The configuration space of a triangular robot is $SE(2) = \mathbb{R}^2 \times S^1$. Using the same input obstacle set as before, our output is shown in Fig. 6. We show the projection of the configuration space $\mathbb{R}^2 \times S^1$ into $\mathbb{R}^2$, but the color scheme is more complicated. In both examples, we used a randomized expansion strategy.

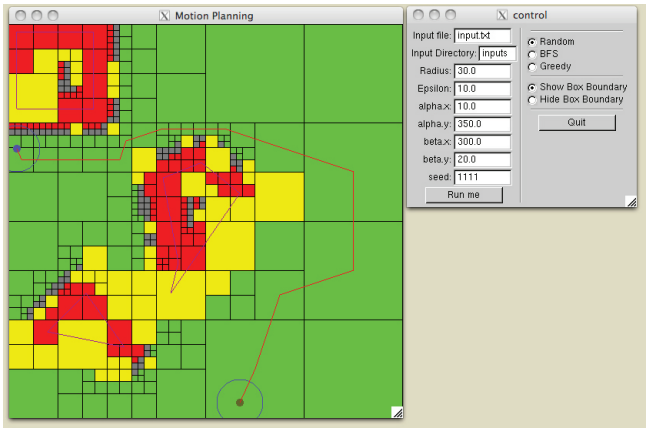

Figure 5: A Subdivision Search for path. Leaf boxes are displayed and color coded (Green=FREE, Red=STUCK, Yellow=large MIXED, Grey=small MIXED).

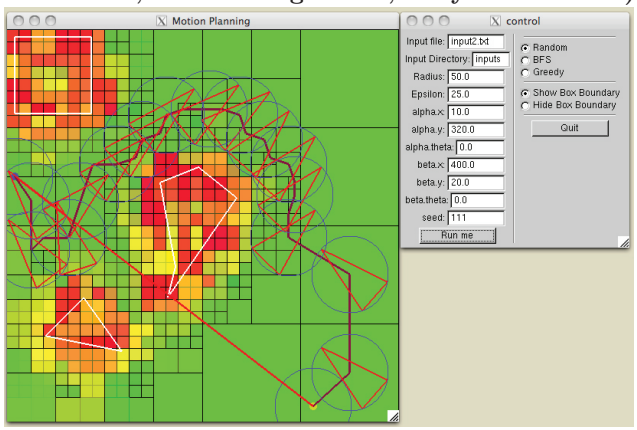

Figure 6: Path for a triangular robot.

Next we show the images of our input datasets used in the experiments in Section 7: bugtrap, input200 and input300, where in each image we show the starting and ending robot configurations indicated by blue circles/triangles (connected by a straight line). The obstacle-polygon edges are shown in white (Fig. 7) or in blue (Figs. 8-9). The paths found by our GBF search strategy are also shown (no path found in Fig. 7). We can see that the triangles may overlap, which can be handled by our approach easily.

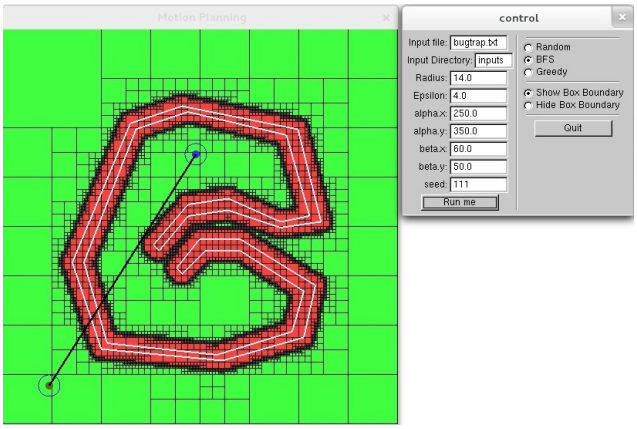

Figure 7: Bugtrap.

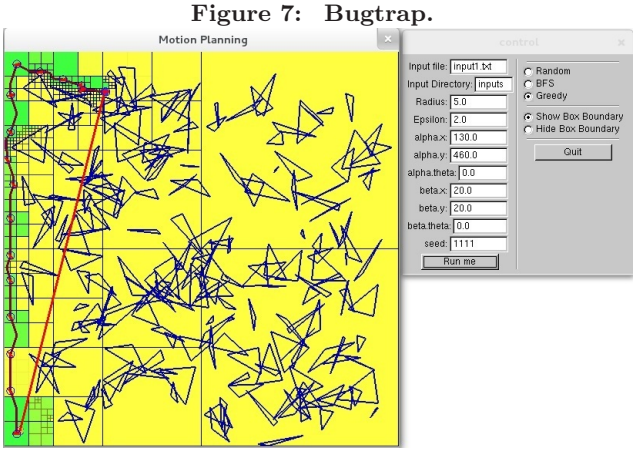

Figure 8: Input200: 200 random triangles.

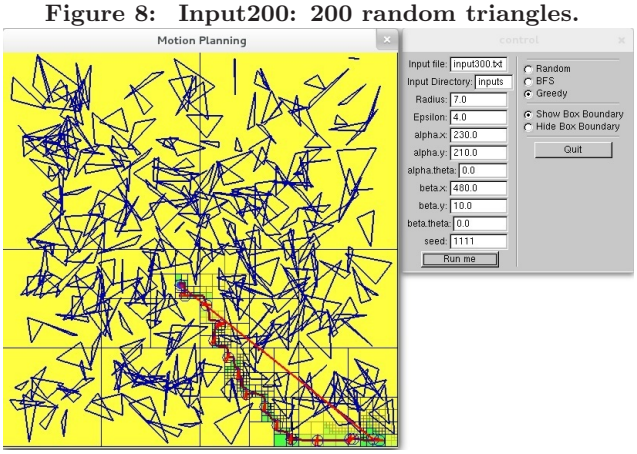

Figure 9: Input300: 300 random triangles.