Resolution-Exact Planner for a 2-link Planar Robot using Soft Predicates

by

Zhongdi Luo

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

Jan 2014

_____ _____

Acknowledgements

I wish to express the deepest appreciation to, first and foremost, my advisor Professor Chee Yap for the persistent help. The works in this thesis are mostly joint work with him. Without his guidance this thesis would not have been possible. His wisdom, generosity and patience taught me innumerable lessons not only of the academic research but also of the life creed.

Besides my advisor, I am also very grateful to Professor Ernest Davis who was willing to read my thesis on a short notice.

I also want to thank all the professors during my graduate studies. Without their classes I would not learn enough preliminaries to enter this research area.

Last, but not least, I would like to thank my wife, Xinting and my parents for their support and love during the past few years.

This thesis is based on the following 2 papers: *Z.Luo, Y-J. Chiang, J.-M. Lien, and C. Yap. Resolution exact algorithms for link robots, 2014. Submitted, SoCG'14* and *Z.Luo and C. Yap. Resolution-Exact Planner for Non-Crossing 2-Link Robot, 2014. Submitted, IROS'14*.

I am very thankful for the support of the Computer Science Department, by awarding me the Master's Dissertation Fellowship, 2013.

This work is partially supported by NSF grant CCF-0917093.

Abstract

Motion planning is a major topic in robotics. It frequently refers to motion of a robot in a $\mathbb{R}^2$ or $\mathbb{R}^3$ world that contains obstacles. Our goal is to produce algorithms that are practical and have strong theoretical guarantees. Recently, a new framework **Soft Subdivision Search** (SSS) was introduced to solve various motion planning problems. It is based on *soft predicates* and a new notion of correctness called *resolution-exactness*. Unlike most theoretical algorithms, such algorithms can be implemented without exact computation.

In this thesis we describe a detailed, realized algorithm of SSS for a 2-link robot in $\mathbb{R}^2$.

We prove the correctness of our predicates and also do experimental study of several strategies to enhance the basic SSS algorithm. In particular, we introduce a technique called T/R Splitting, in which the splittings of the rotational degrees of freedom are deferred to the end. The results give strong evidence of the practicability of SSS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motion Planning Problem

A central problem of robotics is motion planning [2, 9, 11, 4]. The earliest interest in this problem was in the artificial intelligence community where it was known as the find-path problem. There are three main approaches to algorithmic motion planning: exact, sampling and subdivision approaches [11]. The exact approach have been developed by Computational Geometers [6] and in computer algebra [1]. The correct implementation of exact methods is highly non-trivial because of numerical errors. The sampling approach is best represented by Probabilistic Roadmap (PRM) [8] and its many variants (see [15]). It is the dominant paradigm among roboticists today. Subdivision is one of the earliest approaches to motion planning [3]. Recently, we revisit the subdivision approach from a theoretical standpoint [16, 18]. The present work continues this line of development.

Worst-case complexity in motion planning are too pessimistic and ignored issues like large constants, correct implementation of primitives, and adaptive behavior. Roboticists prefer to use empirical criteria to measure the success of

various methods. For instance, Choset et al [4, p. 197-198, Figure 7.1] noted that sampling methods (but not exact or subdivision methods) "can handle" planning problems for a certain 10 DOF planar robot. It roughly means that sampling methods for this robot could terminate in reasonable time on reasonable examples. Of course, this is a far cry from the usual theoretical guarantees of performance. In contrast, there are not only no exact algorithms for this robot, but the usual exact technique of building the entire configuration space is a non-starter. Likewise, standard subdivision methods would fail badly on so many degrees of freedom.

It is suggested [4, p. 202] that the current state of the art in PRM "can handle" 5 to 12 DOF robots; subdivision methods are said to reach medium DOF robots (say, 4 to 7 DOFs).

According to Zhang[19], there are no known good implementations of exact motion planners for more than 3-DOF. On the other hand, their work [19] shows that subdivision methods "can handle" 4 to 6 DOF robots, including the gear robot with complex geometry.

In the face of such empirical evidence, is it possible to design theoretical algorithms that are practical and which roboticists want to implement? Our answer is yes, but we do not come down on the side of exact algorithms. The three approaches (sampling, subdivision and exact) provide increasingly stronger algorithmic guarantees. So the above empirical observations about their relative abilities is not surprising. Barring other issues, one might think we should use the strongest algorithmic method that "can handle" a given robot. Nevertheless, subdivision [16, 18]is preferable to both exact and sampling methods for two fundamental reasons. First, robotic systems are inherently approximate. Exact

computation makes little sense in such a setting, while subdivision appear to naturally support approximation. But to systematically design approximate algorithms, we need a replacement for the standard exact model. The notion of **soft-predicates** is introduced as the basis of an approximate computational model. Second, the difficulty of sampling methods with the **halting problem** (seen as the ultimate form of "narrow passage problem" in sampling) is a serious one. Intuitively, researchers realize that subdivision can overcome this (e.g., [19]), but there are pitfalls in formulating the solution: the usual notion of "resolution completeness" is vague about what a subdivision planner must do if there is NO PATH: one solution may re-introduce the halting problem, while another solution might require exact predicates. To avoid the horns of this dilemma, we bring in the concept of **resolution-exactness**. Taken together, soft predicates and resolution-exactness, get rid of the halting problem of exact computation.

Such algorithms promise to recover all the practical advantages of the PRM framework, but with stronger theoretical guarantees. But many challenges lie ahead to realize these goals. We need to test some of the conventional wisdom of roboticists cited above. Is it really true that subdivision is inherently less efficient than sampling methods? This is suggested by the state-of-art techniques, but we do not see an inherent reason. Is randomness the real source of power in sampling methods? There is some debate among roboticists on this point (cf. LaValle et al [10] and Hsu et al [7]). For subdivision algorithms, we feel that the current limit of 6 DOF is an interesting barrier to cross.

1.2 New ideas.

With the foundation of resolution-exactness and soft predicates in place [16, 18], we need to develop practical techniques for designing such algorithms. Here we focus on the class of articulated robots.

(1) Soft-predicates for link robots. In the presence of subdivision, exact predicates can be replaced by suitable approximations[16]. soft-predicates can exploit a wide variety of techniques that trade-off ease of implementation against efficiency. Here, we introduce the notion of **forbidden angles** for link robots with length. Our find it is a practical way to handle rotational splittings.

(2) We invent a "T/R Splitting" method based on splitting translational and rotational degrees of freedom in different phases. Consider a freely translating k-link planar robot with $k+2$ degrees of freedom. The naive subdivision would split each box into 2^{k+2} children; already for $k = 2$ or 3, the cost is too high to be practical. An idea [16] is to consider two regimes: configuration boxes are originally in the "large regime" in which we only split the translational DOF. When the boxes are sufficiently small, in the "small regime", we split the angular DOF. But this idea only delays the eventual 2^{k+2}-way splits. Our new idea is that we can do the angular split only *once*, at the level just before the leaves.

(3) Extensions: A remarkable property of subdivision, and unlike exact algorithms, is the flexibility we have in extending its predicates. For instance, a extension is that our 2-Link robot can be easily extended to a k-spider robot. It would not affect the performance too much because of our T/R technique.

(4) Taking a leaf from the successful PRM framework, we formulated [18] the **Soft Subdivision Search** (SSS) Framework. This allows a wide variety of algorithms to be designed in a flexible, plug-and-play framework. In this paper, we explore some global search techniques in SSS, using obstacle features as our principle guide. These are ongoing extensions to our current implementation.

(5) We implemented a 2-link robot (with 4 DOF) in `C++`, and our experiments are encouraging: *our planner can solve a wide range of non-trivial instances in real time. Unlike sampling-based planners, we can terminate quickly in case of NO-PATH.*

Chapter 2

Preliminaries

A typical motion planning problem [11] is to produce a continuous motion that connects a start configuration α and a goal configuration β, while avoiding collision with known obstacles Ω. The robot and obstacle geometry is described in $\mathbb{R}^k$ ($k = 2, 3$).

Let R_0 be a fixed robot living in $\mathbb{R}^k$. It defines a configuration space $C_{space} = C_{space}(R_0)$. We may assume $C_{space}(R_0) \subseteq \mathbb{R}^d$ if R_0 has d degrees of freedom. For any obstacle set $\Omega \subseteq \mathbb{R}^k$, we obtain a corresponding free space $C_{free} = C_{free}(\Omega) \subseteq C_{space}$.

Thus the input of the problem is

$$I = (\Omega, \alpha, \beta, B_0) \tag{2.1}$$

where $\Omega \subseteq \mathbb{R}^k$ is a polyhedral set, $\alpha, \beta \in C_{space}$ are start and goal configurations, and $B_0 \subseteq C_{space}$ is a region-of-interest.

We want to find a path in $B_0 \cap C_{free}$ from α to β; return NO-PATH if no such path exists.

¶1. Fundamentals of Our Subdivision Approach. The following three

concepts are given in details in [16, 18]:

- **Resolution-exactness**: this is our replacement for a standard concept in the subdivision literature called "resolution completeness": 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".

- **Soft Predicates**: we are interested in predicates that classify boxes. Let $\square\mathbb{R}^d$ be the set of closed axes-aligned boxes in $\mathbb{R}^d$. Let $C : \mathbb{R}^d \to \{+1, 0, -1\}$ be an (exact) predicate where $+1, -1$ are called definite values, and 0 the indefinite value. We extend it to boxes $B \in \square\mathbb{R}^d$ as follows: for a definite value $v \in \{+1, -1\}$, $C(B) = v$ if $C(x) = v$ for every $x \in B$. Otherwise, $C(B) = 0$. Call $\widetilde{C} : \square\mathbb{R}^d \to \{+1, 0, -1\}$ a "soft version" of C if whenever $\widetilde{C}(B)$ is a definite value, $\widetilde{C}(B) = C(B)$, and moreover, if for any sequence of boxes B_i ($i \geq 1$) that converges monotonically to a point p, $\widetilde{C}(B_i) = C(p)$ for i large enough.

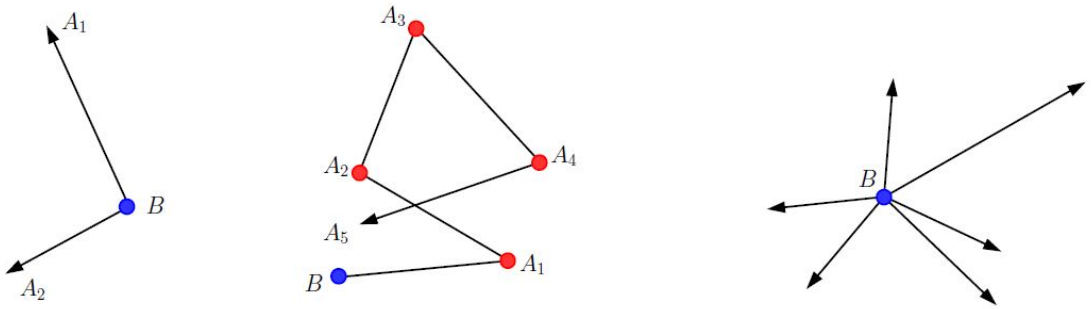
- **Soft Subdivision Search (SSS) Framework**: This is a general framework for a broad class of motion planning algorithms, in the sense that PRM is also such a framework. One must supply a small number of subroutines with fairly general properties in order to derive a specific algorithm. In PRM, one basically needs a subroutine to test if a configuration is free, a method to connect two free configurations, and a method to generate additional configurations. For SSS, we need a predicate to classify boxes in configuration space as `FREE/STUCK/MIXED`, a method to split boxes, and a

method to test if two FREE boxes are connected by a path FREE boxes, and a method to pick MIXED boxes for splitting. The power of such frameworks is that we can explore a great variety of techniques and strategies. This is critical for an area like robotics.

¶2. Link Robots. In our previous work, we focused on rigid robots. Here we look at flexible robots; the simplest such examples are the link robots. Lumelsky and Sun [12] investigated planners for 2-link robots in $\mathbb{R}^2$ and $\mathbb{R}^3$. Sharir and Ariel-Sheffi [14] gave the first exact algorithms for planar k-spider robots.

By a **1-link robot**, we mean a triple $R_1 = (A, B, \ell)$ where A (apex) and B (base) are names for the endpoints of the link, and $\ell > 0$ is the length of the link. Its configuration space is $SE(2) = \mathbb{R}^2 \times S^1$. If $\gamma = (x, y, \theta) \in SE(2)$, then $R_1[\gamma] \subseteq \mathbb{R}^2$ denote the line segment with the B-endpoint at (x, y) and the A-endpoint at $(x, y) + \ell(\cos\theta, \sin\theta)$. Call $R_1[\gamma]$ the **footprint** of R_1 at γ. Also, $A[\gamma], B[\gamma] \in \mathbb{R}^2$ denote the endpoints of $R_1[\gamma]$.

For $k \geq 1$, we define a **k-link robot** R_k recursively: R_1 has been defined, and it has two named points A_0, A_1 (corresponding to the base B and apex A earlier). In general, R_k will have $k+1$ named points: $B = A_0, A_1, \ldots, A_k$. For $k \geq 2$, R_k is a pair (R_{k-1}, L_k) where $L_k = (X_k, A_k, \ell_k)$, X_k is a named point of R_{k-1}, A_k is the new named point, and $\ell_k > 0$ is the length of the kth link. The configuration space of R_k is $C_{space}(R_k) := \mathbb{R}^2 \times (S^1)^k$, with 2 translational DOF's and k rotational DOF's. See for some examples of such robots (k-chains and k-spiders).



(a) R_2 (b) Chain Robot (c) Spider Robot

Figure 2.1: *Some link robots*

We define the **footprint** of R_k: Let $\gamma = (\gamma', \theta_k) \in C_{space}(R_k)$ where $\gamma' = (x, y, \theta_1, \ldots, \theta_{k-1})$. The footprint of the kth link is $L_k[\gamma]$, defined as the line segment with endpoints $X_k[\gamma']$ and $A_k[\gamma] := X_k[\gamma'] + \ell_k(\cos\theta_k, \sin\theta_k)$. The footprint $R_k[\gamma]$ is the union $R_{k-1}[\gamma'] \cup L_k[\gamma]$.

We say γ is **free** if $R_k[\gamma] \cap \Omega = \emptyset$. As usual, $C_{free}(R_k) \subseteq C_{space}(R_k)$ comprise the free configurations. The **clearance** of γ is defined as $C\ell(\gamma) := \mathrm{Sep}(R_k[\gamma], \Omega)$. Here, $\mathrm{Sep}(X, Y) := \inf\{\|x - y\| : x \in X, y \in Y\}$ denotes the **separation** of two Euclidean sets $X, Y \subseteq \mathbb{R}^2$.

¶3. Features-Based Approach. Our computation and predicates are "feature based" whereby the evaluation of box primitives are based on a set $\widetilde{\phi}(B)$ of features associated with the box B.

Given a polygonal set $\Omega \subseteq \mathbb{R}^2$, the boundary $\partial\Omega$ may be subdivided into a unique set of **corners** (points) and **edges** (open line segments), called the **features** of Ω. Let $\Phi(\Omega)$ denote this feature set. Our representation of $f \in \Phi(\Omega)$ ensures this **local property of** f: *for any point q, if f is the closest feature to q, then we can decide if q is inside Ω or not*. To see this, first note that if f is a corner, then q must be outside Ω. So suppose f is a wall. Our representation assigns an

orientation to f such that q is inside Ω iff q lies to the left of of the oriented line through f.

Chapter 3

The T/R Splitting Method

The simplest splitting strategy is to split a box $B \subseteq \mathbb{R}^d$ into 2^d congruent subboxes. This makes sense for a disc robot, but even for the case of $C_{space} = SE(2)$, this strategy is noticeably slow without additional techniques. In [16], The splitting of rotational dimensions is delayed, but the problem of $2^3 = 8$ splits eventually shows up. Here, let's push the delaying idea to the limit: *we would like to split the rotational dimensions only once, at the leaves of the subdivision tree when the translational boxes have radius at most ε.* Moreover, this rotational split can produce arbitrarily many children, depending the number of relevant obstacle features. Intuitively, reducing the translational box down to ε for this technique is not severely inefficient because there are only a 2 DOF for translation. Later, we introduce a modification.

The basis for this approach is a distinction between the translational and rotational components of C_{space}. Note that the rotational component is a subspace of a compact space $(S^1)^k$, and thus it makes sense to treat it differently. Given a box $B \subseteq C_{space}(R_k)$, we write $B = B^t \times B^r$ where $B^t \subseteq \mathbb{R}^2$ and $B^r \subseteq (S^1)^k$ are (respectively) the **translational box** (t-box) and **rotational box** (r-box) corresponding

to B.

For any box $B \subseteq C_{space}(R_k)$, let its **midpoint** $m_B = m(B)$ and **radius** $r_B = r(B)$ refer to the midpoint and radius of its translation part, B^t. Suppose the rotational part of B is given by $B^r = \prod_{i=1}^{k}[\theta_i \pm \delta]$.

Suppose we want to compute a soft predicate $\widetilde{C}(B)$ to classify boxes $B \subseteq C_{space}(R_k)$. Following our previous work [16, 17], we reduce this to computing a feature set $\widetilde{\phi}(B) \subseteq \Phi(\Omega)$. The **feature set** $\widetilde{\phi}(B)$ of B is defined as comprising those features f such that $\mathrm{Sep}(m_B, f) \leq r_B + r_0$ where r_0 is farthest reach of the robot links from its base (i.e., A_0).

We say B is **empty** if $\widetilde{\phi}(B)$ is empty but $\widetilde{\phi}(B_1)$ is not, where B_1 is the parent of B. We may assume the root is never empty. If B is empty, it is easy to decide whether B is `FREE` or `STUCK`: since the feature set $\widetilde{\phi}(B_1)$ is non-empty, we can find the $f_1 \in \widetilde{\phi}(B_1)$ such that $\mathrm{Sep}(m_B, f_1)$ is minimized. Then $\mathrm{Sep}(m_B, f_1) > r_B$, and by the above local property of features, we can decide if m_B is inside or outside Ω. Here then is our (simplified) $Split(B)$ function:

```
Split(B):
    If B is empty,
        Determine if B is free or stuck
    Elif "r(B) > ε"
        t-Split(B)
    Else
        r-Split(B)
```

Here, $t\text{-}Split(B)$ splits only the translational component B (the rotational component remains the full space, $B^r = (S^1)^k$). Similarly, $r\text{-}Split(B)$ splits only B^r and leaves B^t intact. The details of $t\text{-}Split(B)$ are more interesting, and is

taken up in the next chapter.

¶4. Modified T/R Strategy. A possible modification to this T/R strategy is to replace the criterion "$r(B) > \varepsilon$" of $Split(B)$ by "$r(B) > \varepsilon$ and $|\widetilde{\phi}(B)| \geq c$", for some (small) constant c. For instance if $|\widetilde{\phi}(B)| = 2$, we might be in a corridor region and it seems a good idea to start to split the angles. The problem with this variation is that the r-$Split(B)$ gives only an approximation of the possible rotational freedom in B; if no path is found, we may have to split B^t again, in order to apply r-$Split$ to the children of B. This may render it slower than the simple T/R strategy. As our experiments show, a choice like $c = 4$ is a good default.

Chapter 4

Soft Predicate for Rotational Degrees of Freedom

We design the rotational $r\text{-}Split(B)$ routine. Recall that this amounts to splitting B^r but leaving B^t intact. We first assume simple case where R_k is a k-spider. In this case, each link of the robot is independent, so it suffices to consider the case of one link (R_1). For simplicity, assume $B^r = S^1$. If this link has length $\ell > 0$, then $r\text{-}Split(B)$ splits the full circle S^1 into a union of free angular intervals. The number of such free angular ranges is equal to the number of features in $\widetilde{\phi}(B)$ within distance ℓ from $m(B)$.

Use the following convention for closed angular ranges: if $0 \leq \alpha_1 < \alpha_2 < 2\pi$, then let

$[\alpha_1, \alpha_2] := \{\alpha : \alpha_1 \leq \alpha \leq \alpha_2\}$ and $[\alpha_2, \alpha_1] := \{\alpha : 0 \leq \alpha \leq \alpha_1 \text{ or } \alpha_2 \leq \alpha < 2\pi\}$. In any case, if $[\alpha, \alpha']$ is an angular range, we call α (resp., α') the **left stop** (resp., **right stop**) of the angular range.

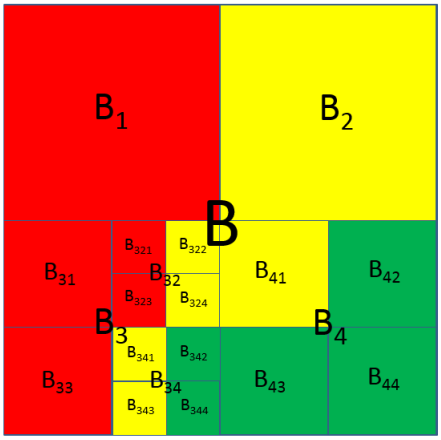


Figure 4.1: *Example of a t-Split process*

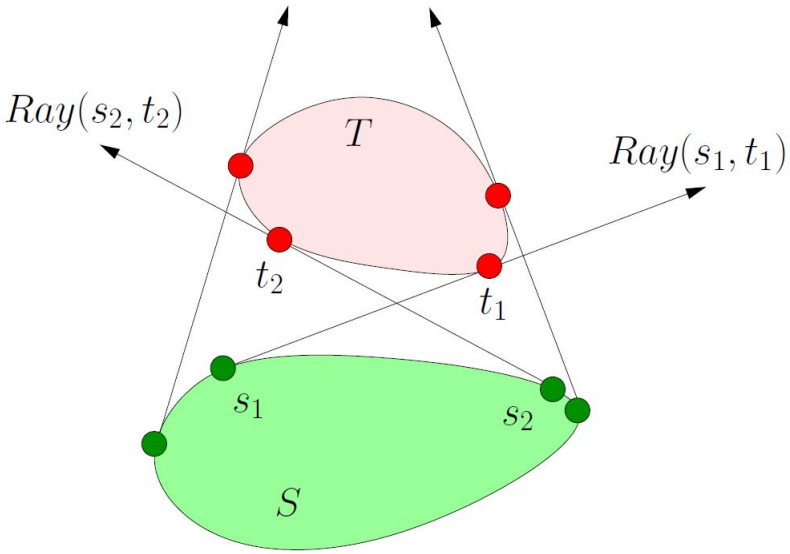


Figure 4.2: *Common tangent rays method used in r-Split*

For $s, t \in \mathbb{R}^2$, let $Ray(s,t)$ denote the ray originating at s and passing through t, and let $\theta(s,t) \in [0, 2\pi) = S^1$ denote its orientation. As usual, let the positive x- and y-axes have orientations 0 and $\pi/2$, respectively. If $S, T \subseteq \mathbb{R}^2$ are sets, let $Ray(S,T) = \{Ray(s,t) : s \in S, t \in T\}$. For $\ell > 0$, define **length-limited** (or

ℓ-limited) **forbidden range** of S, T to be

$$\text{Forb}_\ell(S,T) := \{\theta(s,t) : s \in S, t \in T, \|s-t\| \leq \ell\}.$$

If $S \cap T$ is non-empty, then $\text{Forb}_\ell(S,T) = S^1$. Hence we will assume $S \cap T = \emptyset$. We may also assume S, T are closed convex sets. To understand this forbidden range, we initially set $\ell = \infty$, and simply write $\text{Forb}(S,T)$ for $\text{Forb}_\infty(S,T)$. Call $Ray(s,t) \in Ray(S,T)$ a **common tangent ray** if the line through $Ray(s,t)$ is tangential to S and to T. Such a ray is **separating** if S and T lie on different sides of the line through $Ray(s,t)$. If S, T are not singletons, then there are four common tangent rays, and exactly two of them are separating. We call a separating common tangent ray a **left stop** (resp., a **right stop**) of (S,T) if S lies to the right (resp., left) of the ray. Now it can be verified that $\theta(S,T) = [\theta(s_1,t_1), \theta(s_2,t_2)]$ where $Ray(s_1,t_1)$ and $Ray(s_2,t_2)$ are the left and right stops of (S,T), as illustrated in Figure 4.2.

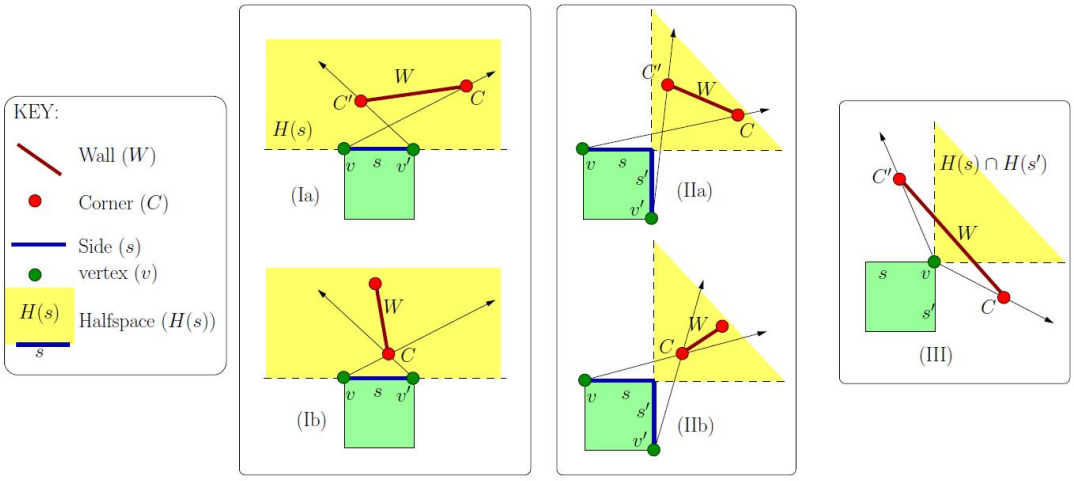


Figure 4.3: *Forbidden Range* $\text{Forb}(B,W)$ *between box* B *and wall* W

We apply these observations to the case where S is a translational box B^t and T is a wall W. If s is a side of B^t, let $H(s)$ denote the closed half-space bounded by s and that has empty intersection with the interior of B^t. Up to symmetry, there are three cases as seen in Figure 4.3:

(I) B^t has a side s such that $W \subseteq H(s)$.

(II) B^t has two consecutive sides s and s' such that $W \subseteq H(s) \cap H(s')$.

(III) B^t has two consecutive sides s and s' such that W crosses $H(s) \cap H(s')$.

We can now easily compute the forbidden range (refer to Figure 4.3):

$$\mathrm{Forb}(B^t, W) = \begin{cases} [\theta(v, C), \theta(v', C')] & \text{if Case (Ia) or (IIa),} \\ [\theta(v, C), \theta(v', C)] & \text{if Case (Ib) or (IIb),} \\ [\theta(v, C), \theta(v, C')] & \text{if Case (III).} \end{cases} \tag{4.1}$$

Then we shall advance to the length-limited forbidden range.

Our goal is to determine $\mathrm{Forb}_\ell(B^t, W)$ where B^t is a translational box and W a wall. We may assume that W does not intersects B^t, because otherwise $\mathrm{Forb}_\ell(B^t, W) = S^1$. It does not appear easy to convincingly enumerate the full range of possibilities for $\mathrm{Forb}_\ell(B^t, W)$ without insight. The first idea as discussed in the text is to focus on the sets $\mathrm{Forb}(B^t, W)$ which is not ℓ-limited. In this appendix, we extend this analysis by decomposing such sets into simpler ones for which the introduction of ℓ is easy to analyze. This is captured by the following lemma:

LEMMA 1. *The set of ℓ-limited forbidden angles $\mathrm{Forb}_\ell(B^t, W)$ can be expressed as the union of at most three sets of the form*

- $\text{Forb}_\ell(v, W)$ *where* v *is a corner vertex of* B^t*, or*
- $\text{Forb}_\ell(s, C)$ *where* s *is a side of* B^t *and* C *is a corner of* W*.*

Proof. We use the cases in the formula (4.1) for $\text{Forb}(B^t, W)$ (please refer to Figure 4.3 for notation):

CASE (I) In CASE (I), the wall W lies in the half-space $H(s)$ for some side s of B^t. We now characterize the distinction between SUBCASES (Ia) and (Ib): let z denote the intersection of the line through W and line through s. If z lies outside s, then we are in SUBCASE (Ia); otherwise we are in SUBCASE (Ib). The situation where z is undefined because W and s are parallel will be treated under SUBCASE (Ia) below.

First consider SUBCASE (Ia) where C, C' are distinct corners of W. Note that $\text{Forb}(B^t, W) = [\theta(v, C), \theta(v', C')]$ can be written as the union of two angular ranges,

$$\text{Forb}(B^t, W) = \text{Forb}(s, C') \cup \text{Forb}(v, W). \tag{4.2}$$

But it could also be written as

$$\text{Forb}(B^t, W) = \text{Forb}(s, C) \cup \text{Forb}(v', W). \tag{4.3}$$

How should we choose between these two representations of $\text{Forb}(B^t, W)$? Recall that SUBCASE (Ia) is characterized by the fact that intersection point z lies outside s; wlog, assume that z lies to the left of s as in .

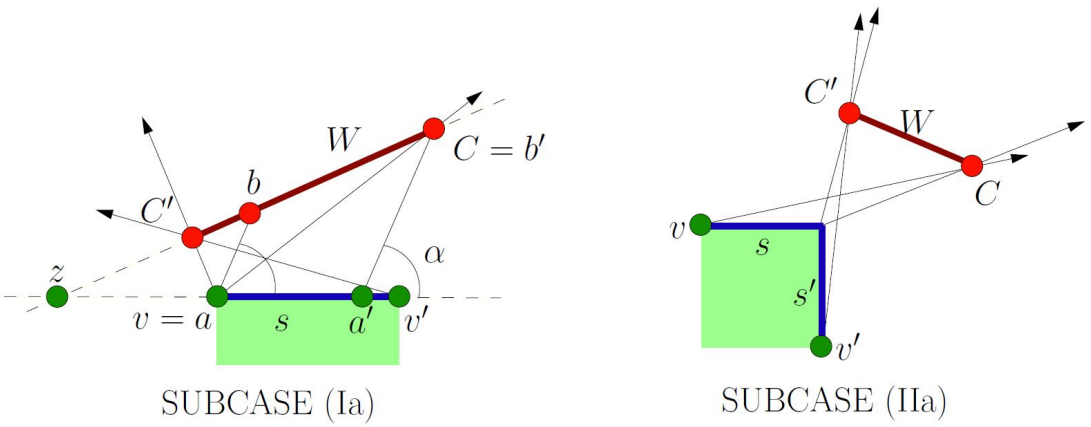


Figure 4.4: *Length-limited Forbidden Zone Analysis*

Suppose $\alpha \in \text{Forb}(B^t, W)$. Then (4.2) implies that there exists a pair

$$(a, b) \in (s \times C') \cup (v \times W)$$

such that $\theta(a, b) = \alpha$. Similarly, (4.2) implies that there exists a pair

$$(a', b') \in (s \times C) \cup (v' \times W)$$

such that $\theta(a, b) = \alpha$. One such angle is illustrated in with $(a, b) = (v, b)$ and $(a', b') = (a', C)$.

It is not hard to verify that the geometry of this subcase implies

$$\|a - b\| \leq \|a' - b'\|.$$

It follows that

$$\alpha \in \text{Forb}_\ell(B^t, W) \iff \alpha \in \text{Forb}_\ell(s, C') \cup \text{Forb}_\ell(v, W).$$

In other words, the representation (4.2) (not (4.3)) extends to the ℓ-limited forbidden angles:

$$\text{Forb}_\ell(B^t, W) = \text{Forb}_\ell(s, C') \cup \text{Forb}_\ell(v, W). \tag{4.4}$$

Note that in case W and s are parallel, both representations (4.2) and (4.3) are equally valid.

It remains to treat SUBCASE (Ib), we have $C = C'$ and so the preceding argument reduces to $\text{Forb}_\ell(B^t, W) = \text{Forb}_\ell(C, s)$.

CASE (II) First consider SUBCASE (IIa) where C, C' are distinct corners of W. The analysis of CASE (Ia) can be applied twice to this case, yielding

$$\text{Forb}_\ell(B^t, W) = \text{Forb}_\ell(v, W) \cup \text{Forb}_\ell(s, C') \cup \text{Forb}_\ell(s', C'). \tag{4.5}$$

For SUBCASE (IIb), we have $C = C'$ and so $\text{Forb}_\ell(v, W)$ can be omitted. Thus $\text{Forb}_\ell(B^t, W) = \text{Forb}_\ell(s, C) \cup \text{Forb}_\ell(s', C)$.

CASE (III) This is simply

$$\text{Forb}_\ell(B^t, W) = \text{Forb}_\ell(v, W). \tag{4.6}$$

Q.E.D.

This lemma shows how to reduce $\text{Forb}_\ell(B, W)$ to the special cases of $\text{Forb}_\ell(v, W)$ and $\text{Forb}_\ell(s, C)$. It remains to show how to determine these sets. But this is

easy: if $D_\ell(v)$ denotes the disc centered at v of radius ℓ, then

$$\left.\begin{aligned} \mathrm{Forb}_\ell(v, W) &= \mathrm{Forb}(v, D_\ell(v) \cap W), \\ \mathrm{Forb}_\ell(s, C) &= \mathrm{Forb}(D_\ell(C) \cap s, C). \end{aligned}\right\}$$

This concludes our analysis of $\mathrm{Forb}_\ell(B, W)$. The proof of our lemma also contain the necessary information for implementing $\mathrm{Forb}_\ell(B, W)$.

Now we are ready to describe the *r-Split*(B) operator: Consider the set $S^1 \setminus \left(\bigcup_i \mathrm{Forb}_\ell(B^t, W_i)\right)$ where W_i range over all walls with at least one corner in $\widetilde{\phi}(B)$. Write this set as the union of disjoint angular ranges

$$\Theta(B) := A_1 \cup A_2 \cup \cdots \cup A_k. \tag{4.7}$$

Then *r-Split*(B) returns the set of boxes $B^t \times A_i$ (for $i = 1, \ldots, k$). The union of these boxes is denoted $\bigcup$*r-Split*(B).

We address correctness of this method. The following lemma is about convergence and effectivity. Clearly, $\bigcup$*r-Split*$(B) \subseteq B \cap C_{free}$. But how good is $\bigcup$*r-Split*(B) as an approximation of $B \cap C_{free}$? This is the question about effectivity of our approximation, and is answered in part(ii) of this lemma.

Lemma 2.
(i) Let $(B_1, B_2, \ldots)$ be a sequence of boxes in C_{space} where $B_i = B_i^t \times S^1$, and B_i^t converges to a point p as $i \to \infty$. Then $\bigcup$r-Split(B_i) converges to the set $(p \times S^1) \cap C_{free}$, i.e., the free configurations with the base at p.
(ii) Let $B = B^t \times S^1$. If $\gamma \in B$ has clearance $C\ell(\gamma) > 2 \cdot r(B)$, then $\gamma \in \bigcup$r-Split(B)

Proof. (i) is immediate. To see (ii), let $\gamma = (p, \theta) \in B$. We prove the contra-positive. Suppose $\gamma \notin \bigcup$*r-Split*(B). Then there is some $p' \in B^t$ such

that $\gamma' = (p', \theta)$ is not free. But $\text{Sep}(R_1[\gamma], R_1[\gamma']) \leq 2 \cdot r(B^t)$. This implies $C\ell(\gamma) \leq 2 \cdot r(B^t) = 2 \cdot r(B)$. **Q.E.D.**

THEOREM 3. *Assume the T/R method for splitting and r-Split is implemented exactly in our SSS Algorithm for a spider robot R_k. Then we obtain an resolution-exact planner for R_k.*

The proof follows the general approach in [16, 18]. Of course, we have no intention of implementing *r-Split* exactly. Instead, if we implement *r-Split*(B) using a conservative approximation with absolute error bounded by $r(B)$, then we obtain a corresponding resolution-exact algorithm (but with larger constant K).

Chapter 5

Experimental Results

We have implemented in `C/C++` the planner for 2-link robots as described in this paper. The experiments are conducted on a laptop with OSX 10.8, Intel Core i7-3610QM CPU and 8GB of RAM.

Our source code and datasets are freely distributed with the `Core Library`[1]. The parameter settings for our experiments on a variety of obstacle inputs are encoded as targets in a `Makefile`. Thus users can easily reproduce our results and conduct further experiments. Here we show the performance on our example targets. Each obstacle set is stored in own text file, and is represented by a set of polygons (not necessarily disjoint) in a square of dimension 512 x 512. Most of the obstacles sets were specially designed for their interest and challenge for our planners. For instance, eg5 involves a double bugtrap and eg300 involves 300 triangles generated at random. Images of our obstacle sets are shown in Figs. 1-7 in the Appendix.

For each obstacle set, Table 5.1 shows two statistics from running our planner: total running time and the total number of tree boxes created. Each run has the

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

parameters $(\ell_1, \ell_2, S, \epsilon, x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$ where ℓ_1, ℓ_2 are the lengths of the 2 links, $S \in \{B, D, G\}$ indicates[2] the search strategy (B = *Breadth First Search* (BFS), D = *Distance + Size*, G = *Greedy Best First* (GBF)), $(x_1, y_1, \alpha_1, \alpha_2)$ is the start configuration and $(x_2, y_2, \beta_1, \beta_2)$ is the goal configuration. In Table 5.1, we pick two variants of T/R splitting: "Simple T/R" means applying *r-Split* when the box size is $< \epsilon$, and "Modified T/R" means applying *r-Split* when the feature set size is small enough (controlled by the parameter c mentioned at the end of Section 3). The choice $c = 4$ is used here. We see that GBF and "Distance + Size" are comparable to each other, and always faster than BFS.

Obstacle Set $(x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$	robot $(\ell_1, \ell_2, S, \epsilon)$	Modified T/R ($c = 4$) time (ms)	Modified T/R ($c = 4$) boxes	Simple T/R time (ms)	Simple T/R boxes	Performance Improvement
eg1	(50,80,**G**,4)	**155.6**	**8232**	158.8	8514	2.0%
(Barrier Environment)	(50,80,**D**,4)	175.3	10886	176.7	10042	0.8%
(195,340,0,150,260,80,70,120)	(50,80,**B**,4)	413.1	29615	403.0	28802	-9.3%
	(**20**,**10**,**G**,4)	25.9	2604	30.3	3985	14.5%
eg2	(85,80,**G**,4)	372.8	23803	482.6	33199	-22.7%
(8-way corridor)	(85,80,**D**,4)	318.6	21400	**316.4**	**20060**	-0.7%
(216,297,115,155,210,220,260,200)	(85,80,**B**,4)	577.8	53393	485.4	48851	-19.0%
eg3	(100,70,**G**,4)	163.5	12530	165.7	12530	1.3%
(8 ways)	(100,70,**D**,4)	136.7	**12005**	**135.4**	**12005**	-0.9%
(262,250,180,270,271,256,90,0)	(100,70,**B**,4)	269.0	21529	274.9	21529	2.1%
eg5	(60,50,**G**,4)	559.1	22781	**532.5**	**22617**	-5.0%
(Double Bugtrap)	(60,50,**D**,4)	625.3	25007	645.1	27185	3.1%
(190,210,180,300,30,30,90,0)	(60,50,**B**,4)	684.0	40007	687.9	39868	0.6%
	(**20**,**20**,**G**,4)	107.5	7637	122.3	10858	12.1%
eg10	(65,80,**G**,4)	99.1	9068	100.8	9068	1.7%
(2 Chambers)	(65,80,**D**,4)	**74.6**	**7416**	75.3	**7416**	0.9%
(425,410,180,270,105,90,90,0)	(65,80,**B**,4)	151.8	15627	147.7	15627	-2.8%
eg300	(40,30,**G**,4)	205.6	**6132**	**201.4**	6133	-2.1%
(300 Triangles)	(40,30,**D**,4)	201.5	6376	201.8	6337	0.1%
(10,400,90,270,330,40,270,0)	(40,30,**B**,4)	2657.3	52318	2414.1	49944	-10.1%

Table 5.1: *Statistics of running our algorithms with* $\epsilon = 4$

In Table 5.2, we fix the other configurations and try to compare the cases with different ϵ. We can see when ϵ gets smaller, more splits are generated and the

[2]Note that a random strategy is never competitive.

calculation gets much slower. According to the result, when we halve ϵ, we get nearly 2X-6X more boxes. It fits the quadruple ratio from translational splittings. The ratio may get worse in Modified T/R since it may suffer a lot by doing useless rotational splittings.

Obstacle Set $(x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$	Robot $(\ell_1, \ell_2, S, \epsilon)$	Modified T/R ($c = 4$) time (ms)	Modified T/R ($c = 4$) boxes	Simple T/R time (ms)	Simple T/R boxes	Ratio by Boxes (using ϵ=1 as base)	
eg1	(50,80,D,**4**)	**175.3**	**10886**	176.7	10042	39.22	12.67
(Barrier Environment)	(50,80,D,**2**)	958.1	71596	651.0	35229	5.96	3.61
(195,340,0,150,260,80,70,120)	(50,80,D,**1**)	5293.2	426975	2506.2	127265	1.0	1.0
eg2	(85,80,D,**4**)	335.4	21400	**323.4**	**20060**	7.99	8.52
(8-way corridor)	(85,80,D,**2**)	624.0	47662	623.9	47662	3.59	3.59
(216,297,115,155,210,220,260,200)	(85,80,D,**1**)	2234.2	170901	2225.0	170901	1.0	1.0
eg3	(100,70,D,**4**)	136.7	**12005**	**135.4**	**12005**	19.49	19.49
(8 ways)	(100,70,D,**2**)	591.5	55227	588.6	55227	4.24	4.24
(262,250,180,270,271,256,90,0)	(100,70,D,**1**)	2505.9	233951	1460.0	233951	1.0	1.0
eg5	(60,50,D,**4**)	**625.3**	**25007**	645.1	27185	35.66	15.63
(Double Bugtrap)	(60,50,D,**2**)	2432.1	102984	2517.2	107113	8.66	3.97
(190,210,180,300,30,30,90,0)	(60,50,D,**1**)	17497.7	891809	9740.9	425011	1.0	1.0
eg10	(65,80,D,**4**)	**74.6**	**7416**	75.3	**7416**	14.94	14.94
(2 Chambers)	(65,80,D,**2**)	300.3	29165	299.7	29165	3.80	3.80
(425,410,180,270,105,90,90,0)	(65,80,D,**1**)	1212.3	110779	1198.2	110771	1.0	1.0
eg12	(45,45,D,**4**)	363.9	**17375**	**359.3**	17494	16.10	14.33
(Maze)	(45,45,D,**2**)	1288.8	65979	1304.4	65751	4.24	3,81
(405,400,180,270,105,60,0,180)	(45,45,D,**1**)	5341.7	279818	4972.4	250772	1.0	1.0
eg300	(40,30,D,**4**)	**201.5**	6376	201.8	**6337**	11.56	11.60
(300 Triangles)	(40,30,D,**2**)	594.5	19304	573.7	19273	3.82	3.81
(10,400,90,270,330,40,270,0)	(40,30,D,**1**)	2223.0	73719	2240.5	73509	1.0	1.0

Table 5.2: *Statistics of running our algorithms with different ϵ*

In Table 5.3, we compare the configurations with different c from 1 to 6. According to the result, the best c may vary. It's hard to get a constant suitable c for each case.

Obstacle Set $(x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$	Robot $(\ell_1, \ell_2, S, \epsilon, c)$	Modified T/R time (ms)	Modified T/R boxes	Simple T/R time (ms)	Simple T/R boxes
eg1	(50,80,D,4,**1**)	173.0	10042	176.7	10042
(Barrier Environment)	(50,80,D,4,**2**)	178.8	10042	-	-
(195,340,0,150,260,80,70,120)	(50,80,D,4,**3**)	174.8	10269	-	-
	(50,80,D,4,**4**)	175.3	10886	-	-
	(50,80,D,4,**5**)	199.6	11773	-	-
	(50,80,D,4,**6**)	294.7	17220	-	-
eg2	(85,80,D,4,**1**)	299.2	20060	300.4	20060
(8-way corridor)	(85,80,D,4,**2**)	301.5	20100	-	-
(216,297,115,155,210,220,260,200)	(85,80,D,4,**3**)	305.0	20576	-	-
	(85,80,D,4,**4**)	318.6	21400	-	-
	(85,80,D,4,**5**)	347.0	24017	-	-
	(85,80,D,4,**6**)	330.1	23717	-	-
eg3	(100,70,D,4,**1**)	137.4	12005	135.4	12005
(8 ways)	(100,70,D,4,**2**)	137.5	12005	-	-
(262,250,180,270,271,256,90,0)	(100,70,D,4,**3**)	136.8	12005	-	-
	(100,70,D,4,**4**)	136.7	12005	-	-
	(100,70,D,4,**5**)	136.9	12005	-	-
	(100,70,D,4,**6**)	137.4	12005	-	-
eg5	(60,50,D,4,**1**)	620.9	26790	645.1	27185
(Double Bugtrap)	(60,50,D,4,**2**)	628.9	26617	-	-
(190,210,180,300,30,30,90,0)	(60,50,D,4,**3**)	619.0	25384	-	-
	(60,50,D,4,**4**)	625.3	25007	-	-
	(60,50,D,4,**5**)	619.9	25101	-	-
	(60,50,D,4,**6**)	731.7	34662	-	-
eg10	(65,80,D,4,**1**)	73.6	7416	75.3	7416
(2 Chambers)	(65,80,D,4,**2**)	74.0	7416	-	-
(425,410,180,270,105,90,90,0)	(65,80,D,4,**3**)	73.4	7416	-	-
	(65,80,D,4,**4**)	74.6	7416	-	-
	(65,80,D,4,**5**)	74.5	7416	-	-
	(65,80,D,4,**6**)	78.6	7948	-	-
eg12	(45,45,D,4,**1**)	359.3	17494	359.3	17494
(Maze)	(45,45,D,4,**2**)	358.6	17376	-	-
(405,400,180,270,105,60,0,180)	(45,45,D,4,**3**)	357.7	17352	-	-
	(45,45,D,4,**4**)	363.9	17375	-	-
	(45,45,D,4,**5**)	361.0	17383	-	-
	(45,45,D,4,**6**)	356.3	17338	-	-
eg300	(40,30,D,4,**1**)	195.7	6337	201.8	6337
(300 Triangles)	(40,30,D,4,**2**)	194.9	6337	-	-
(10,400,90,270,330,40,270,0)	(40,30,D,4,**3**)	195.5	6376	-	-
	(40,30,D,4,**4**)	201.5	6376	-	-
	(40,30,D,4,**5**)	195.3	6393	-	-
	(40,30,D,4,**6**)	192.5	6311	-	-

Table 5.3: *Statistics of running our algorithms with different c*

Chapter 6

Non-Crossing 2-Link Robot

In this chapter we analyze a special kind of 2-link robot called **Non-Crossing** robot. To introduce this concept, we address the following phenomena at first: in the screen shot of Figure 6.1, we show two configurations of the 2-link robot inside an (inverted) T-room environment. Let α (respectively, β) be the start (goal) configuration as indicated[1] by the double (single) circle. There are obvious paths from α to β whereby the robot origin moves directly from the start to goal positions and simultaneously, the link angles monotonically adjust themselves, as illustrated in Figure 6.3(a). However, such paths require the two links to cross each other. To achieve a "non-crossing" path from α to β, the robot origin must initially move away from the goal configuration towards the T-junction first, in order to maneuver the two links into the appropriate relative order before it can move toward the goal configuration. Such a non-crossing path is shown in Figure 6.3(b). We find such paths with a subdivision algorithm; Figure 6.2 illustrates the subdivision of the underlying configuration space (the scheme is explained below).

[1]Our images have color: the double circle is cyan and single circle is magenta. The robot links are colored blue (ℓ_1) and red (ℓ_2), respectively.

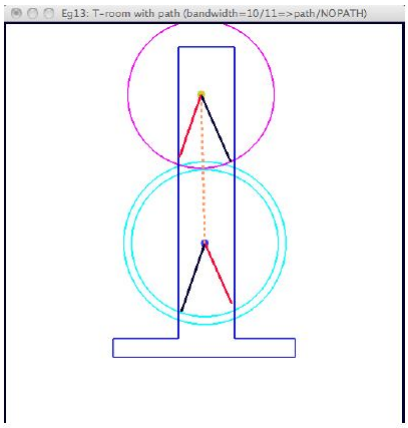


Figure 6.1: *T-Room: start and goal configurations*

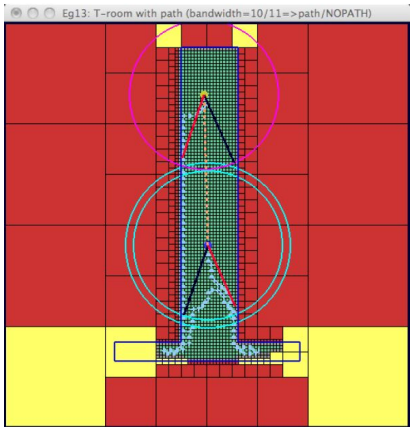


Figure 6.2: *T-room: Subdivision*

Here we show how to construct a practical and theoretically-sound planner for a non-crossing 2-link robot. Our planner may (but not always) suffer a loss of efficiency when compared to the self-crossing 2-link planner (see [13]). Nevertheless, it gives real-time performance for a variety of non-trivial obstacle environments such as illustrated in Figure 6.4 (200 randomly generated triangles), Figure 6.5(a) (Double Bug-trap (cf. [p. 181, Figure 5.13][11]), Figure 6.5(b) (Maze).

For example, Figure 6.4(a) is an environment with 200 randomly generated

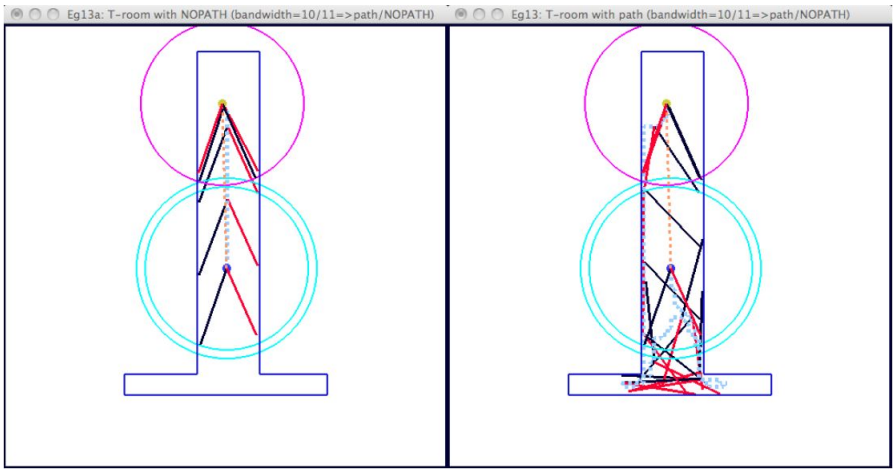


Figure 6.3: *T-room: Left side is the self-crossing solution, Right side is the non-crossing solution.*

triangles, and a path is found with $\varepsilon = 4$. If we use $\varepsilon = 5$, then it returns NO-PATH as shown in Figure 6.4(b). This NO-PATH declaration guarantees that there is no path with clearance $> K\varepsilon$ (for some constant K).

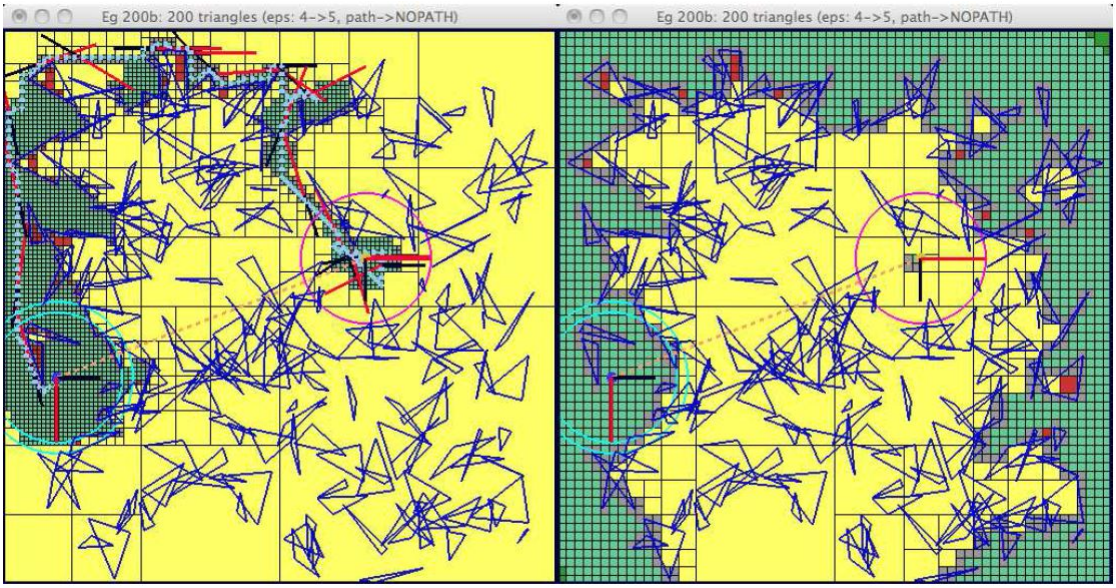
6.1 Configuration Space of Non-Crossing Robot.

The **self-crossing configuration space** of R_2 is

$$C_{space} = \mathbb{R}^2 \times \mathcal{T} \tag{6.1}$$

where $\mathcal{T} = S^1 \times S^1$ is the torus. The configuration of link robots is treated in Devadoss and O'Rourke [5, chap. 7]. Consider a configuration $\gamma = (x, y, \theta_1, \theta_2) \in C_{space}$ where θ_i $(i = 1, 2)$ is the orientation of the i-th link (see Figure 2.1(a)). When $\theta_1 = \theta_2$, we say the configuration is **self-crossing**; otherwise it is **non-crossing**. Let

$$\Delta := \left\{(\theta, \theta) : \theta \in S^1\right\}, \qquad \mathcal{T}_\Delta := \mathcal{T} \setminus \Delta.$$



(a) Path found with $\varepsilon = 4$ (b) NO-PATH found with $\varepsilon = 5$

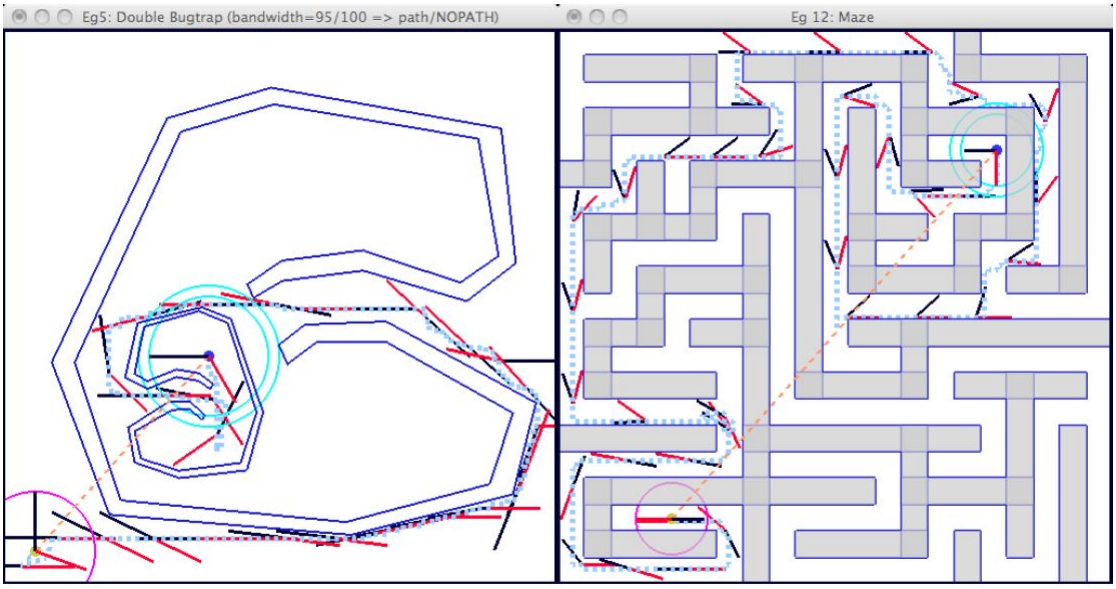
Figure 6.4: *200 Random Triangles.*

Also, let $\mathcal{T}_< := \{(\theta, \theta') \in \mathcal{T} : \theta < \theta'\}$ and $\mathcal{T}_> := \{(\theta, \theta') \in \mathcal{T} : \theta > \theta'\}$. We are interested in planning the motion of R_2 in the **non-crossing configuration space**,

$$C_{space}^{\Delta} := \mathbb{R}^2 \times \mathcal{T}_{\Delta}. \tag{6.2}$$

Note that Δ is a closed curve in $\mathcal{T}$. In $\mathbb{R}^2$, a closed curve will disconnect the plane into two connected components. But the curve Δ does not disconnect $\mathcal{T}$. To see this, consider the plane model of $\mathcal{T}$ represented by a square with opposite sides identified as shown in Figure 6.6. Each of the sets $\mathcal{T}_<$ and $\mathcal{T}_>$ are themselves connected; moreover, any $\alpha \in \mathcal{T}_>$ and $\beta \in \mathcal{T}_<$ are path-connected in $\mathcal{T}_{\Delta}$ (as illustrated in Figure 6.6).

Let us be specific about how to interpret the parameters of C_{space}. The robot R_2 has three named points A_0, A_1, A_2 (see [13]), shown in Figure 2.1(a), where A_0 is the robot joint (or origin).The **footprint** of these points at a configuration



(a) Double Bugtrap (b) Maze

Figure 6.5: *(a) Double Bugtrap, (b) Maze.*

$\gamma = (x, y, \theta_1, \theta_2)$ are given by

$$\begin{aligned} A_0[\gamma] &:= (x, y), \\ A_1[\gamma] &:= (x, y) + \ell_1(\cos\theta_1, \sin\theta_1), \\ A_2[\gamma] &:= (x, y) + \ell_2(\cos\theta_2, \sin\theta_2). \end{aligned}$$

Let $R_2[\gamma] \subseteq \mathbb{R}^2$ denote the **footprint** of R_2 at γ, defined as the union of the line segments $[A_0[\gamma], A_1[\gamma]]$ and $[A_0[\gamma], A_2[\gamma]]$.

6.2 Subdivision for Non-Crossing 2-Link Robot

Suppose we already have a resolution-exact planner for a self-crossing 2-Link robot. We now describe a simple transformation to convert it into a resolution-exact planner for a non-crossing 2-Link robot.

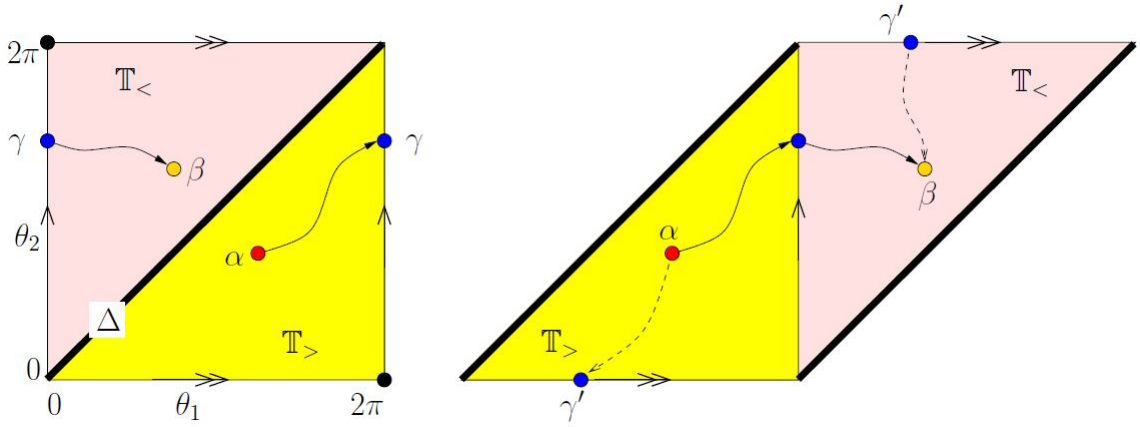


Figure 6.6: *Paths in $\mathcal{T}_\Delta$ from $\alpha \in \mathcal{T}_>$ to $\beta \in \mathcal{T}_<$*

6.2.1 Subdivision of Boxes

In this paper, we are interested in a slight generalization of such boxes.

By a **box** (or d-box) of dimension $d \geq 1$ we mean a set of the form $B = \prod_{i=1}^{d} I_i$ where $d \geq 1$ and each I_i is a closed interval of $\mathbb{R}$ or S^1 of positive length. Such boxes are natural for doing subdivision in configuration spaces of the form $\mathbb{R}^k \times (S^1)^{d-k}$. For our 2-link robots, $d = 4$ and $k = 2$. The configuration space for a submarine or helicopter might be regarded as $\mathbb{R}^3 \times S^1$.

For $i = 1, \ldots, d$, we have the notion of ***i*-projection** and ***i*-coprojection** of d-boxes:

- (Projection) $\texttt{Proj}_i(B) := \prod_{j=1, j\neq i}^{d} I_j$ is a $(d-1)$ dimensional box.
- (Co-Projection) $\texttt{Coproj}_i(B) := I_i$ is the ith interval of B.

We also define the **indexed Cartesian product** $\otimes_i$ via the identity

$$B = \texttt{Proj}_i(B) \otimes_i \texttt{Coproj}_i(B).$$

Let $j = -1, 0, 1, \ldots, d$. Two boxes B, B' of dimension $d \geq 1$ are said to be

j-**adjacent** if $\dim(B \cap B') = j$. Note that B and B' are (-1)-adjacent means they are disjoint. When $i = d - 1$, we simply say the boxes are **adjacent**, denoted $B :: B'$; when $i = d$, we say they are **overlapping**, denoted $B \circ B'$. The following is immediate:

LEMMA 4. *Let B, B' be boxes of dimension $d \geq 1$.*

- *If $d = 1$ then $B :: B'$ iff $|B \cap B'| \in \{1, 2\}$.*
- *If $d > 1$ then $B :: B'$ iff $(\exists i = 1, \ldots, d)$ such that*

$$\texttt{Proj}_i(B) \circ \texttt{Proj}_i(B) \quad \wedge \quad \texttt{Coproj}_i(B) :: \texttt{Coproj}_i(B').$$

6.2.2 Boxes for Non-Crossing Robot.

Our basic idea for representing boxes in the non-crossing configuration space C^{Δ}_{space} is to write it as a pair $(B, \texttt{XT})$ where $\texttt{XT} \in \{\texttt{LT}, \texttt{GT}\}$, and B is a box in self-crossing configuration space C_{space}. The pair $(B, \texttt{XT})$ represents the set $B \cap (\mathbb{R}^2 \times \mathcal{T}_{\texttt{XT}})$ (with the identification $\mathcal{T}_{\texttt{LT}} = \mathcal{T}_<$ and $\mathcal{T}_{\texttt{GT}} = \mathcal{T}_>$). It is convenient to call $(B, \texttt{XT})$ an X**-box** since they are no longer "boxes" in the usual sense.

As in [13], we may write B as the Cartesian product of a translational box B^t and a rotational box B^r: $B = B^t \times B^r$ where $B^t \subseteq \mathbb{R}^2$ and $B^r \subseteq \mathcal{T}$. Thus, $B^r = \Theta_1 \times \Theta_2$ where $\Theta_1, \Theta_2 \subseteq S^1$ are angular intervals. We denote (closed) angular intervals by $[s, t]$ where $s, t \in [0, 2\pi]$ and using the interpretation

$$[s, t] := \begin{cases} \{\theta : s \leq \theta \leq t\} & \text{if } s < t, \\ [s, 2\pi] \cup [0, t] & \text{if } s \geq t. \end{cases}$$

In particular, $[s,s] = [s,t] \cup [t,s] = S^1$. An angular interval $[s,t]$ that[2] contains a neighborhood of $0 = 2\pi$ is said to be **wrapping**. Also, call $B^r = \Theta_1 \times \Theta_2$ wrapping if either Θ_1 or Θ_2 is wrapping.

Given any B^r, we can decompose the set $B^r \cap \mathcal{T}_\Delta$ into the union of two subsets $B^r_{\mathtt{LT}}$ and $B^r_{\mathtt{GT}}$, where $B^r_{\mathtt{XT}}$ denote the set $B^r \cap \mathcal{T}_{\mathtt{XT}}$. In case B^r is non-wrapping, this decomposition has the nice property that each subset $B^r_{\mathtt{XT}}$ is connected. For this reason, we prefer to work with non-wrapping boxes. Initially, the box $B^r = \mathcal{T}$ is wrapping. The initial split of $\mathcal{T}$ should be done in such a way that the children are all non-wrapping: the "natural" (quadtree-like) way to split $\mathcal{T}$ into four congruent children has[3] this property. Thereafter, subsequent splitting of these non-wrapping boxes will remain non-wrapping.

Of course, $B^r_{\mathtt{XT}}$ might be empty, and this is easily checked: say $\Theta_i = [s_i, t_i]$ $(i = 1, 2)$. Then $B^r_<$ is empty iff $t_2 \leq s_1$. and $B^r_>$ is empty iff $s_2 \geq t_1$. Moreover, these two conditions are mutually exclusive.

We now modify the algorithm of [13] as follows: as long as we are just splitting boxes in the translational dimensions, there is no difference. When we decide to split the rotational dimensions, we use the T/R splitting method of [13], but each child is further split into two X-boxes annotated by `LT` or `GT` (they are filtered out if empty). We build the connectivity graph G (see Appendix 2) with these X-boxes as nodes. This ensures that we only find non-crossing paths. Our algorithm inherits resolution-exactness from the original self-crossing algorithm.

[2]Wrapping intervals are either equal to S^1 or has the form $[s,t]$ where $s > t$, $s \neq 2\pi$ and $t \neq 0$

[3]This is not a vacuous remark – the quadtree-like split is determined by the choice of a "center" for splitting. To ensure non-wrapping children, this center is necessarily $(0,0)$ or equivalently $(2\pi, 2\pi)$. Furthermore, our T/R splitting method (to be introduced) does not follow the conventional quadtree-like subdivision at all.

6.3 Extension to Diagonal Band

The diagonal Δ is a curve with no width. We now want to fatten Δ into a band $\Delta(\kappa)$ of **bandwidth** $\kappa \geq 0$. For this extension, we use the intrinsic Riemannian metric on S^1: the distance between $\theta, \theta' \in S^1$ is given by

$$d(\theta, \theta') = \min\left\{|\theta - \theta'|, 2\pi - |\theta - \theta'|\right\}.$$

where we may assume $\theta, \theta' \in [0, 2\pi]$. Fix $0 \leq \kappa < \pi$. Then

$$\Delta(\kappa) := \left\{(\theta, \theta') \in \mathcal{T} : d(\theta, \theta') \leq \kappa\right\}.$$

Thus the original diagonal line is $\Delta(0) = \Delta$. The non-crossing configuration space is now

$$C_{space}^{\Delta(\kappa)} = \mathbb{R}^2 \times (\mathcal{T} \setminus \Delta(\kappa)).$$

This extension is very useful in applications. For example, the T-room example (Figures 6.1–6.2) uses $\kappa = 10^\circ$. Moreover, if we set $\kappa = 11^\circ$, then there is NO-PATH. It is not surprising that as κ is increased, we may no longer be able to find a path. But somewhat surprisingly, our experiments (see Table I below) show that increasing κ may also speed up the search for a path.

The predicate $\texttt{isBoxEmpty}(B^r, \kappa, \texttt{XT})$ which returns true iff $(B^r_{\texttt{XT}}) \cap \mathcal{T}_{\Delta(\kappa)}$ is empty is useful in implementation. It has a simple expression when restricted to non-wrapping translational box B^r:

Lemma 5.

Let $B^r = [a, b] \times [a', b']$ be a non-wrapping box.

(a) $\texttt{isBoxEmpty}(B^r, \kappa, \texttt{LT}) = \texttt{true}$ *iff* $\kappa \geq b' - a$ *or* $2\pi - \kappa \leq a' - b$.

(b) $\texttt{isBoxEmpty}(B^r, \kappa, \texttt{GT}) = \texttt{true}$ *iff* $\kappa \geq b - a'$ *or* $2\pi - \kappa \leq a - b'$.

6.4 Implementation and Experiments

Our current implementation is based on machine arithmetic, but it is relatively straightforward to ensure arbitrary precision using bigFloat numbers and "lax comparison" as described in [16].

Table 6.1 compares the performance of the non-crossing planner with the original crossing planner from [13]. Each row of Table I shows two statistics for the self-crossing and non-crossing planners: total running time and the total number of subdivision boxes created. The last column shows the percentage improvement in time for non-crossing over self-crossing.

We use various obstacle sets (named `egX` such as `eg2a`, `eg2b`, `eg5`, etc.). Each run is a row in the Table, and has these parameters $(\ell_1, \ell_2, S, \varepsilon, \kappa)$ where ℓ_i is the length of the i-th link, $S \in \{B, D, G\}$ indicates[4] the search strategy (B = *Breadth First Search* (BFS), D = *Distance + Size*, G = *Greedy Best First* (GBF)). The last parameter $\kappa \in [0, 180)$ is the bandwidth of Δ in degrees. When we run the self-crossing planner the κ parameter is ignored. The parameters for each run are encoded in a Makefile.

Table 6.1 shows that the running time of the non-crossing planner is comparable to that of the self-crossing planner in all the examples (with the exception of the T-room or eg13). Their percentage change is between -44.8% to 11.4%. That is because, although non-crossing planner has some overhead, it also filters out useless splittings earlier for the dead ends. The exceptional case (T-room)

[4] A random strategy is available, but it is never competitive.

is explained by the fact that the non-crossing planner must use a much longer circuitous path.

Table 6.2 shows the sensitivity of finding a path to the link length ℓ_2, and to the bandwidth κ, as ε decreases.

Obstacle Set	Configuration	Self-Crossing		Non-Crossing		Performance
$(x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$	$(\ell_1, \ell_2, S, \epsilon, \kappa)$	time (ms)	boxes	time (ms)	boxes	Improvement
eg2b	(88, 98, D, 2, 79)	1740.1	104663	**1591.9**	**71123**	8.51%
(8-way corridor)	(88, 98, D, 2, 80)	-	-	No Path	No Path	-
(216,297,95,175,210,220,270,190)	(88, 98, D, 2, 30)	-	-	1687.1	101287	3.0%
	(88, 98, D, 2, 5)	-	-	1963.2	129394	-12.8%
eg5	(55, 50, G, 4, 95)	**541.2**	**22243**	542.3	27560	-0.2%
(Double Bugtrap)	(55, 50, G, 4, 100)	-	-	No Path	No Path	-
(190,210,180,300,30,30,90,340)	(55, 50, G, 4, 50)	-	-	613.1	32157	-13.3%
	(55, 50, G, 4, 10)	-	-	730.3	42994	-34.9%
eg8	(30, 25, G, 2, 7)	**31.5**	**2215**	45.6	5214	-44.8%
(Hsu et al. [7])	(30, 25, G, 2, 8)	-	-	No Path	No Path	-
(20,390,0,270,275,180,270,0)	(30, 25, G, 2, 3)	-	-	37.3	3514	-18.4%
eg12	(30, 33, D, 4, 146)	314.4	19953	**283.3**	**15167**	9.9%
(Maze)	(30, 33, D, 4, 147)	-	-	No Path	No Path	-
(375,400,180,0,105,60,0,180)	(30, 33, D, 4, 40)	-	-	360.9	22908	-14.8%
	(30, 33, D, 4, 10)	-	-	410.2	32783	-30.5%
eg13	(94, 85, D, 4, 10)	**3.1**	**616**	98.9	12212	-3090%
(T-Room)	(94, 85, D, 4, 11)	-	-	No Path	No Path	-
(275,230,251,294,252,420,294,251)	(94, 85, D, 4, 5)	-	-	94.9	12068	-2961%
eg300	(40, 30, G, 4, 127)	305.7	8794	**270.8**	**7314**	11.4%
(300 Triangles)	(40, 30, G, 4, 128)	-	-	No Path	No Path	-
(10,400,90,270,270,190,270,90)	(40, 30, G, 4, 40)	-	-	353.6	11284	-15.7%
	(40, 30, G, 4, 10)	-	-	348.4	12113	-14.0%

Table 6.1: *Comparison between Self-Crossing and Non-Crossing.*

Although our current techniques work well for this 4DOF robot, we believe that new techniques are needed to address higher DOF's. We are working on robots in $\mathbb{R}^3$. But even in the plane, real-time performance is easily compromised. For instance, we could clearly extend the current work to non-crossing k-spiders for $k \geq 3$, with $C_{space} = \mathbb{R}^2 \times \mathcal{T}^k$ where $\mathcal{T}^k = (S^1)^k$. We expect to be achieve real-time performance for $k = 3, 4$. However, it is less clear that we can do the same with k-chain robots for $k \geq 3$, crossing or non-crossing.

Obstacle Set $(x_1, y_1, \alpha_1, \alpha_2, x_2, y_2, \beta_1, \beta_2)$	Configuration $(\ell_1, \ell_2, S, \epsilon, \kappa)$	Self-Crossing time (ms)	Self-Crossing boxes	Non-Crossing time (ms)	Non-Crossing boxes	Performance Improvement
eg2a	(85, 80, G, 8, 10)	No Path	No Path	No Path	No Path	-
(8-way corridor)	(85, 80, G, **4**, 10)	459.0	33199	**400.9**	**31390**	12.7%
(216,297,155,115,210,220,260,200)	(85, **92**, G, 4, 10)	No Path	No Path	No Path	No Path	-
	(85, 92, G, **2**, 10)	**2271.8**	**153425**	2402.3	192916	-5.7%
	(85, **99**, G, 2, 10)	No Path	No Path	No Path	No Path	-
	(85, 99, G, **1**, 10)	**5887.4**	**385814**	6190.0	448119	-5.1%
	(85, **100**, G, 1, 10)	No Path	No Path	No Path	No Path	-
eg13	(94, 85, D, 8, 10)	No Path	No Path	No Path	No Path	-
(T-Room)	(94, 85, D, **4**, 10)	**3.1**	**616**	98.9	12212	-3090%
(275,230,251,294,252,420,294,251)	(94, 85, D, 4, **13**)	-	-	No Path	No Path	-
	(94, 85, D, **2**, 13)	**6.2**	**1187**	417.7	47292	-6637%
	(94, 85, D, 2, **14**)	-	-	No Path	No Path	-
	(94, 85, D, **1**, 14)	**9.8**	**1974**	1553.7	184559	-15754%
	(94, 85, D, 1, **15**)	-	-	No Path	No Path	-

Table 6.2: *(a) Eg2a shows the sensitivity to length ℓ_2 as ε changes. (b) Eg13 shows the sensitivity to bandwidth κ as ε changes.*

Chapter 7

Conclusions

We hope that the focus on soft methods will usher in renewed interest in theoretically sound and practical algorithms in robotics, and more generally in Computational Geometry. Our experimental results for link robots offer hopeful signs that this is possible.

Our basic SSS framework (like PRM) is capable of many generalizations for motion planning. One direction is to consider multiple-query models; another is to exploit the stuck boxes for faster termination in case of NO-PATH. Extensions to kinodynamic planning offer a chance at practical algorithms in this important area where no known theoretical algorithms are practical. Much theoretical and complexity analysis remains open.

It is clear that the theory of soft subdivision methods can be generalized and extended to many traditional problems in Computational Geometry[18]. But it also extends to new areas that are currently untouchable by our exact computational models, especially those that are defined by non-algebraic continuous data.

Further work:

- The algorithms of 2-link robot can be adapted to support k-spider robot.
- 2-link robot can be expanded to chain robot with 3 or more links. It will need new soft predicate heuristics to design and implement.

Bibliography

[1] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer, 2nd edition, 2006.

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

[3] R. A. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. 8th Intl. Joint Conf. on Artificial intelligence - Volume 2*, pages 799–806, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[4] 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.

[5] S. L. Devadoss and J. O'Rourke. *Discrete and Computatational Geometry*. Princeton University Press, 2011.

[6] 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.

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

[8] 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.

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

[10] S. LaValle, M. Branicky, and S. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Int'l. J. Robotics Research*, 23(7/8):673–692, 2004.

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

[12] V. Lumelsky and K. Sun. A unified methodology for motion planning with uncertainty for 2d and 3d two-link robot arm manipulators. *Int'l. J. Robotics Research*, 9:89–104, 1990.

[13] Z. Luo, Y.-J. Chiang, J.-M. Lien, and C. Yap. Resolution exact algorithms for link robots, 2014. Submitted, 30th ACM Symp. on Comp. Geom. Preliminary version: 23rd Fall Workshop on Comp. Geom. (FWCG), Oct 25-26, 2013. The City College of New York.

[14] M. Sharir, , and E. Ariel-Sheffi. On the piano movers' problem: Iv. various decomposable two-dimensional motion planning problems. NYU Robotics Report 58, Courant Institute, New York University, 1983.

[15] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, Cambridge, MA, 2005.

[16] C. Wang, Y.-J. Chiang, and C. Yap. On Soft Predicates in Subdivision Motion Planning. In *29th ACM Symp. on Comp. Geom. (SoCG'13)*, pages 349–358, 2013. Full paper was invited and submitted to *Comp.Geom.: Theory & Applic. (CGTA), Special Issue for SoCG '13*.

[17] 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).*, pages 2–16. IEEE, 2012. Invited Talk. June 27-29, 2012, Rutgers University, NJ.

[18] 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). Robotics Science and Systems Conference (RSS 2013), Berlin, Germany, June 27, 2013. Full paper from: http://cs.nyu.edu/exact/papers/.

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

Appendices

APPENDIX 1: Experimental Setup

The following figures (Figs. 1-7) show the GUI interface to our implementation. The right hand control panel allows the user to rerun with new parameters, replay the animation, to replay the splitting of boxes, change ε or the start and goal configurations, choose different global search strategies, modify dimensions of the robot, etc. Subdivision boxes can also be queried.

We have a collection of predefined environments, and for each environment, we selected some interesting parameters and encode them as targets for a Makefile. These targets are named "egX" where X=0,1,2, etc. The examples run in real time, even in case of NO-PATH (a feat that would challenge sampling-based methods). Correctness of the solution is independent of the search strategy. The straightforward randomized strategy tend to be the slowest.

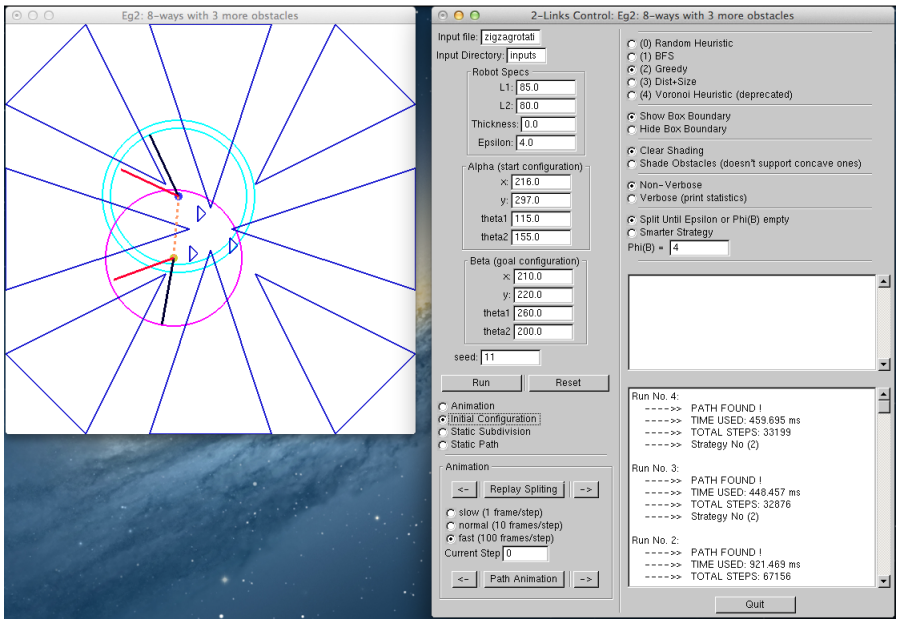


Figure 1: *Start and goal configurations for Obstacle Set eg2.*

The obstacle set has 8 big triangles forming 8-way "corridors", plus 3 small triangles in the center. We also show the starting and ending configurations of the 2-link robot (enclosed in a single circle and double circles respectively).

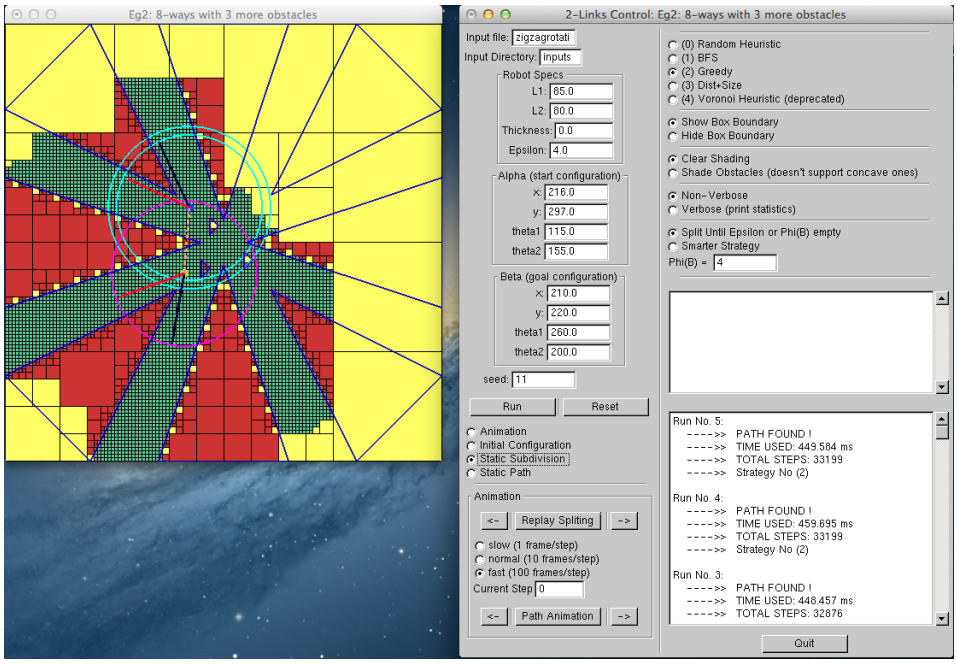


Figure 2: *Final subdivision for Obstacle Set eg2.*

The resulting path is shown, and the leaf boxes obtained during the subdivision search for the path are displayed and color coded.

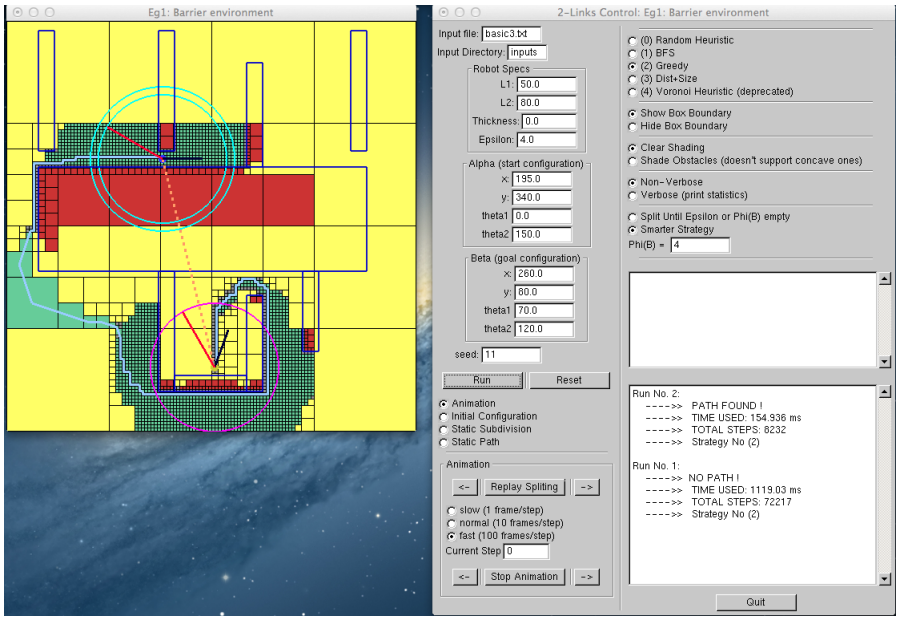


Figure 3: *The eg1 dataset, "barrier"*

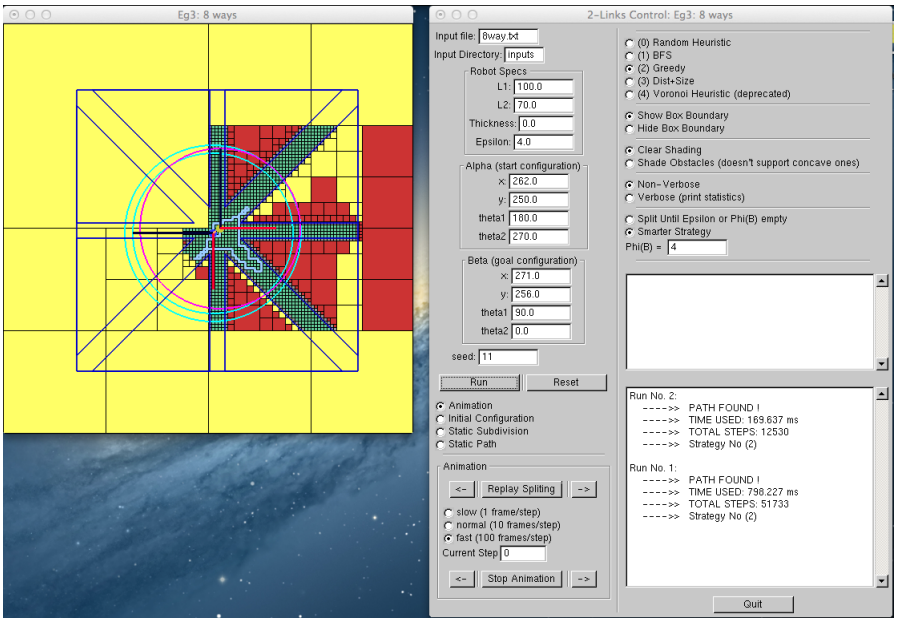


Figure 4: *The eg3 dataset, another version of 8-way "corridors"*

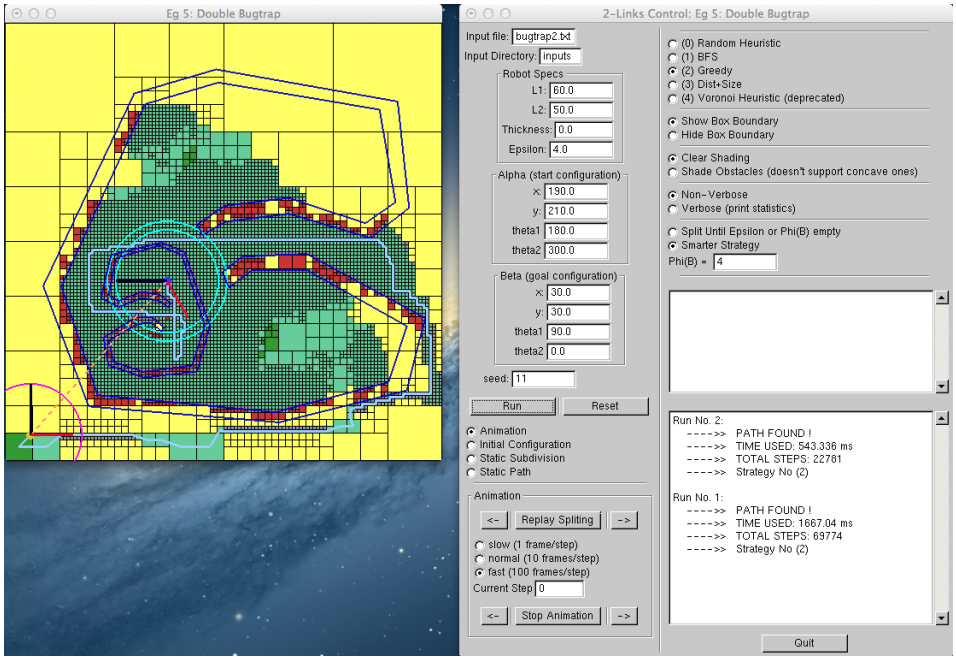


Figure 5: *The eg5 dataset, "double bugtrap"*

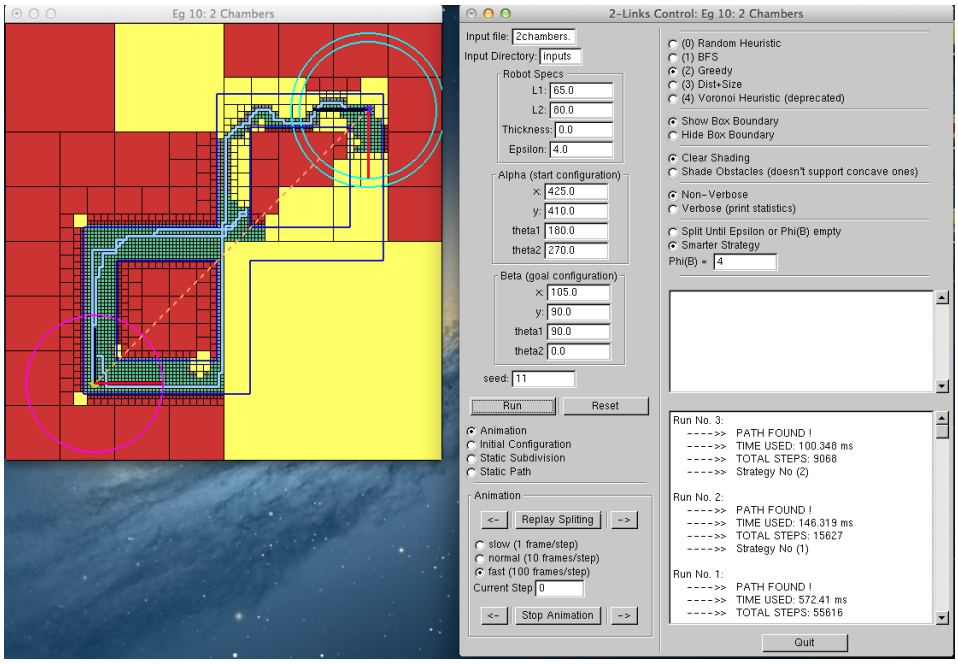


Figure 6: *The eg10 dataset, "2 chambers"*

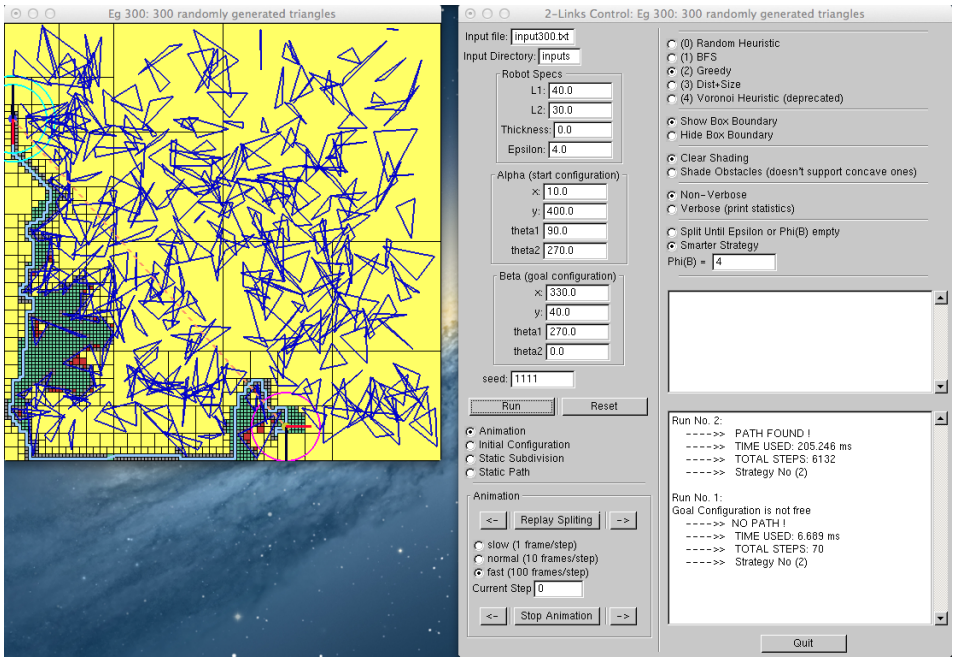


Figure 7: *The eg300 dataset, which has 300 randomly generated triangles*

APPENDIX 2: Workflow of SSS Framework

We describe the main procedure of our algorithm. See [16, 18] for more details.

Our algorithm grows a subdivision tree $\mathcal{T}(B_0)$ rooted at B_0. Each tree node is a box), the set of children of an internal node B forms a subdivision of B. Consequently, the leaves of $\mathcal{T}(B_0)$ forms a subdivision of B_0. The nodes of $\mathcal{T}(B_0)$ are classified as `FREE`, `STUCK`, or `MIXED`, with the requirement that (1) `FREE`-nodes are subsets of $C_{free}(R_0)$, (2) `STUCK`-nodes does not intersect $C_{free}(R_0)$, (3) the internal nodes are `MIXED`.

There is a natural relationship adjacency graph $G(\mathcal{T}(B_0))$ whose nodes are the leaves of $\mathcal{T}(B_0)$ and two nodes are **adjacent** if they share a $(d-1)$-face. The tree $\mathcal{T}(B_0)$ stops growing the moment we discover a `FREE`**-channel** connecting α, β, i.e., a path

$$(B_1, B_2, \ldots, B_m) \tag{1}$$

in the graph $G(\mathcal{T}(B_0))$ where $\alpha \in B_1, \beta \in B_m$ and each B_i is `FREE`. Let $PathP(\alpha, \beta)$ be the predicate that is true iff the `FREE`-channel (1) exists in $G(\mathcal{T}(B_0))$. This predicate is efficiently implemented with standard techniques involving the Union-Find data structure. If $Path(\alpha, \beta)$ holds, there are also standard methods to extract a path in C_{space} from α to β from $G(\mathcal{T}(B_0))$; but here we suppress details of the Union-Find and path extraction subroutines.

Here is the basic SSS algorithm: let $Box_{\mathcal{T}}(\alpha)$ denote the leaf box containing α (similarly for $Box_{\mathcal{T}}(\alpha)$). Note that there are two while-loops before the MAIN LOOP, and these keep splitting $Box_{\mathcal{T}}(\alpha)$ and $Box_{\mathcal{T}}(\beta)$ until they are FREE, or else return NO-PATH.

SSS Framework:

Input: Configurations α, β, tolerance $\varepsilon > 0$, box $B_0 \in \mathbb{R}^d$.

Initialize a subdivision tree $\mathcal{T}$ with root B_0.

Initialize priority queue Q, graph G and union-find data structure.

1. $currentBox \leftarrow Box_{\mathcal{T}}$

While ($currentBox \neq$ FREE)

If radius of $currentBox \leq \varepsilon$, Return(NO-PATH)

Else $childrenBoxes \leftarrow$ Split($currentBox$)

For $childBox \in childrenBoxes$

If $Position(\alpha) \in childBox$, $currentBox \leftarrow childBox$

$startBox \leftarrow currentBox$

2. $currentBox \leftarrow Box_{\mathcal{T}}$

While ($currentBox \neq$ FREE)

If radius of $currentBox \leq \varepsilon$, Return(NO-PATH)

Else $childrenBoxes \leftarrow$ Split($currentBox$)

For $childBox \in childrenBoxes$

If $Position(\beta) \in childBox$, $currentBox \leftarrow childBox$

$goalBox \leftarrow currentBox$

3. Add B_0 into $Q_{\mathcal{T}}$.

4. While ($Find(startBox) \neq Find(goalBox)$) ◁ *MAIN LOOP*

If $Q_{\mathcal{T}}$ is empty, Return(NO-PATH)

$currentBox \leftarrow \mathcal{T}$.GetNext() ◁ *VARIOUS STRATEGIES*

$childrenBoxes \leftarrow$ Split($currentBox$) ◁ *DETERMINE T/R SPLITTING*

For $childBox \in childrenBoxes$

If $childBox$ is FREE

Union $childBox$ with its FreeNeighbors separately.

Add $childBox$ into G.

If $childBox$ is MIXED and radius of $childBox \leq \varepsilon$

Add $childBox$ into $Q_{\mathcal{T}}$.

5. Generate a shortest path from $startBox$ to $goalBox$ using G.