Resolution-Exact Planner for Non-Crossing 2-Link Robot

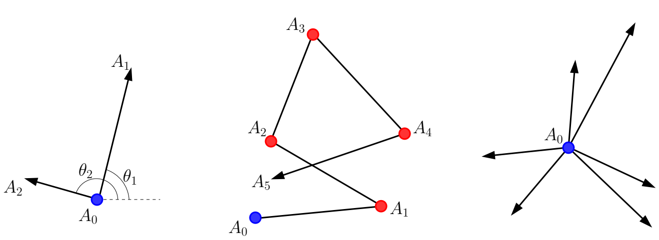
Zhongdi Luo and Chee K. Yap
Department of Computer Science
Courant Institute, NYU
New York, NY 10012, USA
{zl562,yap}@cs.nyu.edu

***Abstract*— We consider the motion planning problem for a 2-link robot amidst polygonal obstacles. The two links are normally allowed to cross each other, but in this paper, we introduce the non-crossing version of this robot. This 4DOF configuration space is novel and interesting.**

Using the recently introduced algorithmic framework of Soft Subdivision Search (SSS), we design a resolution-exact planner for this robot. We introduce a new data structure for representing boxes and doing subdivision in this non-crossing configuration space. Our implementation achieved real-time performance for a suite of non-trivial obstacle sets.

I. INTRODUCTION

Motion planning is one of the key topics of robotics [7], [3]. The main approach to motion planning for almost two decades now is probabilistic sampling such as PRM [6]. In the plane, the simplest example of a flexible or non-rigid robot is the **2-link robot**, $R_2 = R_2(\ell_1, \ell_2)$, with links of positive lengths ℓ_1 and ℓ_2. The two links are connected through a rotational joint A_0 called the **robot origin** as illustrated in Figure 1(a). The 2-link robot is in the intersection of two well-known families of link robots: **chain robots** and **spider robots** (Figure 1(b,c)). See [9] for a definition of **link robots**; there we also discuss link robots with thickness.



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

Fig. 1: Link Robots

We address the following phenomena: in the screen shot of Figure 2, 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 4(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 4(b). We find such paths with a subdivision algorithm; Figure 3 illustrates the subdivision of the underlying configuration space (the scheme is explained below).

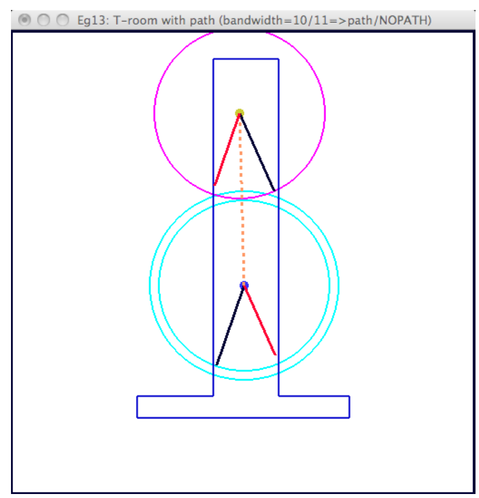


Fig. 2: T-Room: start and goal configurations

This paper shows how to construct a practical and theoretically-sound planner for a non-crossing 2-link robot. To our knowledge, this non-crossing configuration space has not been studied before. Our planner may (but not always) suffer a loss of efficiency when compared to the self-crossing 2-link planner (see [9]). Nevertheless, it gives real-time performance for a variety of non-trivial obstacle environments such as illustrated in Figure 5 (200 randomly generated triangles), Figure 6(a) (Double Bug-trap (cf. [p. 181, Figure 5.13][7]), Figure 6(b) (Maze). Unlike sampling based planners, we do not need any pre-processing arguments, and no special techniques are needed to address the halting or narrow-passage problem. For example, Figure 5(a) is an environment with 200 randomly generated triangles, and a path is found with $\varepsilon = 4$. If we use $\varepsilon = 5$, then it returns NO-PATH as shown in Figure 5(b). This NO-PATH declaration

[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.

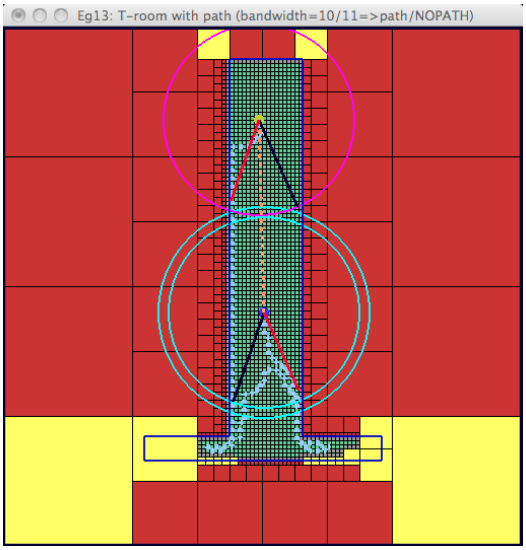
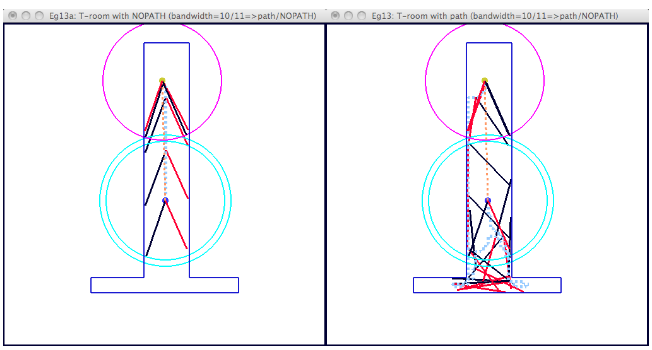


Fig. 3: T-room: Subdivision



(a) Self-crossing path (b) Non-crossing path

Fig. 4: T-Room Environment

guarantees that there is no path with clearance $> K\varepsilon$ (for some constant K).

II. Configuration Space of Non-Crossing Robot.

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

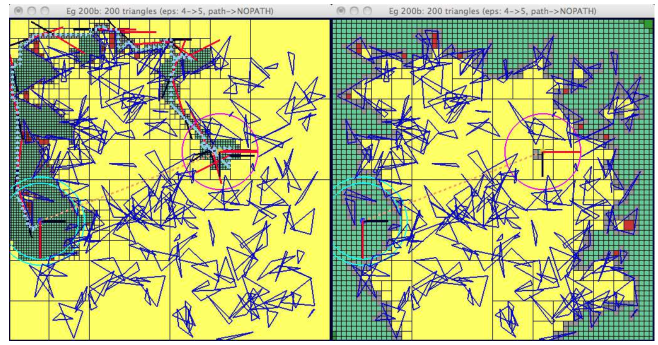
$$C_{space} = \mathbb{R}^2 \times \mathbb{T} \tag{1}$$

where $\mathbb{T} = S^1 \times S^1$ is the torus. The configuration of link robots is treated in Devadoss and O'Rourke [4, 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 1(a)). When $\theta_1 = \theta_2$, we say the configuration is **self-crossing**; otherwise it is **non-crossing**. Let

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

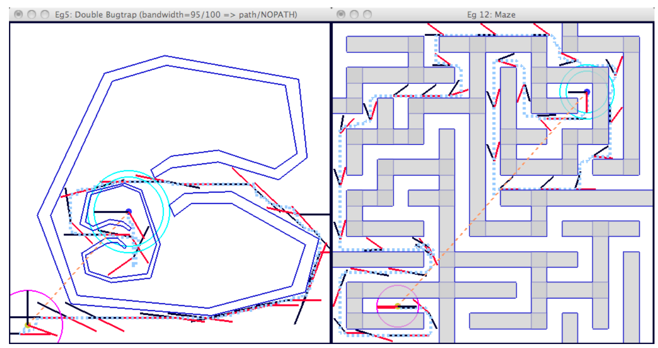
Also, let $\mathbb{T}_< := \{(\theta, \theta') \in \mathbb{T} : \theta < \theta'\}$ and $\mathbb{T}_> := \{(\theta, \theta') \in \mathbb{T} : \theta > \theta'\}$. We are interested in planning the motion of R_2 in the **non-crossing configuration space**,

$$C^\Delta_{space} := \mathbb{R}^2 \times \mathbb{T}_\Delta. \tag{2}$$



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

Fig. 5: 200 Random Triangles.



(a) Double Bugtrap (b) Maze

Fig. 6: (a) Double Bugtrap, (b) Maze.

Note that Δ is a closed curve in $\mathbb{T}$. In $\mathbb{R}^2$, a closed curve will disconnect the plane into two connected components. But the curve Δ does not disconnect $\mathbb{T}$. To see this, consider the plane model of $\mathbb{T}$ represented by a square with opposite sides identified as shown in Figure 7. Each of the sets $\mathbb{T}_<$ and $\mathbb{T}_>$ are themselves connected; moreover, any $\alpha \in \mathbb{T}_>$ and $\beta \in \mathbb{T}_<$ are path-connected in $\mathbb{T}_\Delta$ (as illustrated in Figure 7).

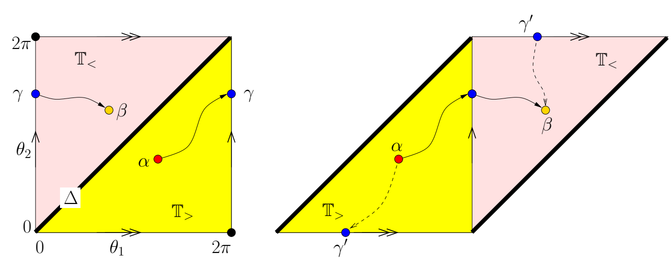


Fig. 7: Paths in $\mathbb{T}_\Delta$ from $\alpha \in \mathbb{T}_>$ to $\beta \in \mathbb{T}_<$

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 [9]), shown in Figure 1(a), where A_0 is the robot joint (or origin).The **footprint** of these points at a configuration $\gamma = (x, y, \theta_1, \theta_2)$ 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 $[A_0[\gamma], A_2[\gamma]]$.

III. Resolution-Exact Planning

The **separation** of two sets $S, T \subseteq \mathbb{R}^2$ is $\mathrm{Sep}(S,T) := \inf\{\|s-t\| : s \in S, t \in T\}$. The **clearance** of $\gamma \in C_{space}$ relative to any set $\Omega \subseteq \mathbb{R}^2$ is $\mathrm{Sep}(R_2[\gamma], \Omega)$, denoted $C\ell(\gamma,\Omega)$ or $C\ell(\gamma)$ when Ω is understood. A configuration γ is **Ω-free** if $C\ell(\gamma,\Omega) > 0$. Let $C_{free}(\Omega) = C_{free}(\Omega; R_2)$ denote the set of Ω-free configurations of R_2. A **Ω-free path** (or simply "path") is a continuous function $\mu : [0,1] \to C_{free}(\Omega; R_2)$; the **clearance** of μ is $\inf\{C\ell(\mu(t),\Omega) : t \in [0,1]\}$. The **basic planning problem** for a robot R is this: *given a polygonal set $\Omega \subseteq \mathbb{R}$, a box $B_0 \subseteq C_{space}(R)$ and $\alpha, \beta \in B_0$, to find any Ω-free path $\mu : [0,1] \to B_0$ with $\mu(0) = \alpha$ and $\mu(1) = \beta$ if any such path exists; otherwise, return NO-PATH if no such path exists.*

To avoid exact computation, we [10], [11] introduced the **resolution-exact planning problem**: *given $\Omega, B_0, \alpha, \beta$ as before, but additionally $\varepsilon > 0$, to find any Ω-free path $\mu : [0,1] \to B_0$ if there exists any path with clearance $K\varepsilon$; and return NO-PATH if there does not exist a path with clearance ε/K.* Here, $K > 1$ is any constant that depends on the algorithm but independent of the inputs $(\Omega, \alpha, \beta, B_0; \varepsilon)$. For simplicity, we do not require that the returned path μ have any specified clearance; in [10] we require μ to have clearance ε/K.

¶1. Soft-Subdivision Search To construct resolution-exact planners, we use the well-known subdivision paradigm [2], [13], [1], [12]. Our subdivision framework for such planners is called **Soft Subdivision Search** (SSS), and exploits the concept of soft predicates. Appendix A reviews these concepts. To get a planner for any specific robot like our 2-link robot, we need three subroutines:

- Soft Predicate $\widetilde{C}$ for classifying boxes: for each box B, $\widetilde{C}(B) \in \{\mathtt{MIXED}, \mathtt{FREE}, \mathtt{STUCK}\}$. Leaf boxes that are `MIXED` and not "ε-small" are placed in a priority queue Q.
- Search Strategy $Q.\mathtt{GetNext}()$ which returns the next box B in Q to be split. There are canonical choices for $Q.\mathtt{GetNext}()$, such as BFS, random choice, various A-star analogues.
- Split Strategy $\mathtt{Split}(B)$: We could split B into 2^d congruent children if B is a d-dimensional box – but this is unlikely to scale for $d > 3$. We may use global strategies that depend on state information and other computed parameters. Following [9], this paper will use the T/R approach.

Figure 3 shows such a subdivision for our 2-link robot. Each box $B \subseteq C_{space}$ is decomposed into the translation and rotational components: $B = B^t \times B^r$ where $B^t \subseteq \mathbb{R}^2$ and $B^r \subseteq \mathbb{T}$. Our display only shows the square B^t but a user could click B^t to read off the corresponding angular ranges of B^r in the panel. Each box B^r is colored red/green/yellow/gray. Red and green indicate `STUCK` and `FREE` boxes. The `MIXED` boxes are colored yellow and gray, depending on whether its radius is at least ε or not. Thus, only yellow boxes are candidates for splitting.

In this paper, we will concentrate on the soft predicate $\widetilde{C}(B)$. The search strategy $Q.\mathtt{GetNext}()$ can be any of the mentioned canonical ones. The split strategy $\mathtt{Split}(B)$ is the T/R strategy from [9]. The idea is that we split the angular range only when a box B has radius $< \varepsilon$, otherwise we only split its translational subbox B^t. Moreover, the splitting of B^r is not based on binary splits, but depends on the geometry of the obstacles.

IV. 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.

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

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

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

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 1: 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

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

¶3. 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 $(B, \mathtt{XT})$ where $\mathtt{XT} \in \{\mathtt{LT}, \mathtt{GT}\}$, and B is a box in self-crossing configuration space C_{space}. The pair $(B, \mathtt{XT})$ represents the set $B \cap (\mathbb{R}^2 \times \mathbb{T}_{\mathtt{XT}})$ (with the identification $\mathbb{T}_{\mathtt{LT}} = \mathbb{T}_<$ and $\mathbb{T}_{\mathtt{GT}} = \mathbb{T}_>$). It is convenient to call $(B, \mathtt{XT})$ an **X-box** since they are no longer "boxes" in the usual sense.

As in [9], 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 $B^r \subseteq \mathbb{T}$. 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

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

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 $B^r \cap \mathbb{T}_\Delta$ into the union of two subsets $B^r_{\mathtt{LT}}$ and $B^r_{\mathtt{GT}}$, where $B^r_{\mathtt{XT}}$ denote the set $B^r \cap \mathbb{T}_{\mathtt{XT}}$. 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 $B^r = \mathbb{T}$ is wrapping. The initial split of $\mathbb{T}$ should be done in such a way that the children are all non-wrapping: the "natural" (quadtree-like) way to split $\mathbb{T}$ 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 $t_2 \le s_1$. and $B^r_>$ is empty iff $s_2 \ge t_1$. Moreover, these two conditions are mutually exclusive.

We now modify the algorithm of [9] 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 [9], 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 A) 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.

V. 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** $\kappa \ge 0$. For this extension, we use the intrinsic Riemannian metric on S^1: the distance between $\theta, \theta' \in S^1$ is given by

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

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

$$\Delta(\kappa) := \{(\theta,\theta') \in \mathbb{T} : d(\theta,\theta') \le \kappa\}.$$

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

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

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

[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.

This extension is very useful in applications. For example, the T-room example (Figures 2–3) uses $\kappa = 10°$. Moreover, if we set $\kappa = 11°$, 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 $\mathtt{isBoxEmpty}(B^r, \kappa, \mathtt{XT})$ which returns true iff $(B^r_{\mathtt{XT}}) \cap \mathbb{T}_{\Delta(\kappa)}$ is empty is useful in implementation. It has a simple expression when restricted to non-wrapping translational box B^r:

Lemma 2:
Let $B^r = [a,b] \times [a',b']$ be a non-wrapping box.
(a) $\mathtt{isBoxEmpty}(B^r, \kappa, \mathtt{LT}) = \mathsf{true}$ iff $\kappa \ge b' - a$ or $2\pi - \kappa \le a' - b$.
(b) $\mathtt{isBoxEmpty}(B^r, \kappa, \mathtt{GT}) = \mathsf{true}$ iff $\kappa \ge b - a'$ or $2\pi - \kappa \le a - b'$.

VI. Implementation and Experiments

We implemented our planner in `C++`, extending our previous work on self-crossing 2-link robots in [9]. Our code, data and experiments are freely distributed[4] with our open source `Core Library`. See Luo's thesis [8] for more examples. The platform for the experiments is a Mac OS X 10.8.3 (Mountain Lion) with a Quad Core Intel Core i7-3610QM Processor, (6MB L3 Cache, up to 3.30 GHz) and 16GB DDR3-1600MHz RAM. 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 [10].

Table I compares the performance of the non-crossing planner with the original crossing planner from [9]. 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[5] 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, but the user may modify these parameters through the GUI interface (see Figure 8).

Table I 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

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

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

dead ends. The exceptional case (T-room) is explained by the fact that the non-crossing planner must use a much longer circuitous path.

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

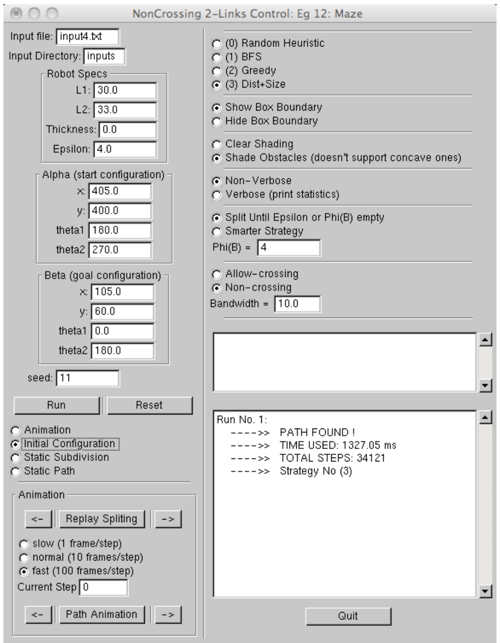


Fig. 8: GUI Interface for Maze Example

VII. Conclusion and Limitations

The introduction of non-crossing flexible robots is novel, and points the way for many similar extensions. Our work is a contribution to the development of practical and theoretically sound subdivision planners [10], [11].

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 $C_{space} = \mathbb{R}^2 \times \mathbb{T}^k$ where $\mathbb{T}^k = (S^1)^k$. 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.

References

[1] M. Barbehenn and S. Hutchinson. Toward an exact incremental geometric robot motion planner. In *Proc. Intelligent Robots and Systems 95.*, volume 3, pages 39–44, 1995. 1995 IEEE/RSJ Intl. Conf., 5–9, Aug 1995. Pittsburgh, PA, USA.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] Z. Luo. Resolution-exact planner for a 2-link planar robot using soft predicates. Master thesis, New York University, Courant Institute, Jan. 2014.

[9] 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.

[10] 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.*

[11] 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/.

[12] 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.

[13] 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 A: Elements of SSS Theory

We review the the notion of soft predicates and how it is used in the SSS Planning Framework. See [10], [11], [9] for more details.

¶4. Soft Predicates. The concept of a "soft predicate" is relative to some exact predicate. Define the exact predicate $C : C_{space} \to \{0, +1, -1\}$ where $C(x) = 0/+1/-1$ (resp.) if configuration x is semi-free/free/stuck. The semi-free configurations are those on the boundary of C_{free}. Call $+1$ and -1 the **definite values**, and 0 the **indefinite value**. Extend the definition to any set $B \subseteq C_{space}$: for a definite value v, define $C(B) = v$ iff $C(x) = v$ for all x. Otherwise, $C(B) = 0$. Let $\square(C_{space})$ denote the set of d-dimensional boxes in C_{space}. A predicate $\widetilde{C} : \square(C_{space}) \to \{0, +1, -1\}$ is a **soft version of** C if it is conservative and convergent. **Conservative** means that if $\widetilde{C}(B)$ is a definite value, then $\widetilde{C}(B) = C(B)$. **Convergent** means that if for any sequence $(B_1, B_2, \ldots)$ of boxes, if $B_i \to p \in C_{space}$ as $i \to \infty$, then $\widetilde{C}(B_i) = C(p)$ for i large enough. To achieve resolution-exact algorithms, we must ensure $\widetilde{C}$ converges quickly in this sense: say $\widetilde{C}$ is **effective** if there is a constant $\sigma > 1$ such if $C(B)$ is definite, then $\widetilde{C}(B/\sigma)$ is definite.

¶5. The Soft Subdivision Search Framework. An SSS algorithm maintains a subdivision tree $\mathcal{T} = \mathcal{T}(B_0)$ rooted at a given box B_0. Each tree node is a subbox of B_0. We assume a procedure $\mathtt{Split}(B)$ that subdivides a given leaf box B into a bounded number of subboxes which becomes the children of B in $\mathcal{T}$. Thus B is "expanded" and no longer a leaf. For example, $\mathtt{Split}(B)$ might create 2^d congruent subboxes as children. Initially $\mathcal{T}$ has just the root B_0; we grow $\mathcal{T}$ by repeatedly expanding its leaves. The set of leaves

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

TABLE I: Comparison between Self-Crossing and Non-Crossing.

Obstacle (input)	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 (8-way corridor)	(85, 80, G, 8, 10)	No Path	No Path	No Path	No Path	-
	(85, 80, G, **4**, 10)	459.0	33199	**400.9**	**31390**	12.7%
	(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 (T-Room)	(94, 85, D, 8, 10)	No Path	No Path	No Path	No Path	-
	(94, 85, D, **4**, 10)	**3.1**	**616**	98.9	12212	-3090%
	(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 II: (a) Eg2a shows the sensitivity to length ℓ_2 as ε changes. (b) Eg13 shows the sensitivity to bandwidth κ as ε changes.

of $\mathcal{T}$ at any moment constitute a subdivision of B_0. Each node $B \in \mathcal{T}$ is classified using a soft predicate $\widetilde{C}$ as $\widetilde{C}(B) \in \{\texttt{MIXED}, \texttt{FREE}, \texttt{STUCK}/\} = \{0, +1, -1\}$. Only `MIXED` leaves with radius $\geq \varepsilon$ are candidates for expansion. We need to maintain three auxiliary data structures:

- A priority queue Q which contains all candidate boxes. Let $Q.\texttt{GetNext}()$ remove the box of highest priority from Q. The tree $\mathcal{T}$ grows by splitting $Q.\texttt{GetNext}()$.
- A **connectivity graph** G whose nodes are the `FREE` leaves in $\mathcal{T}$, and whose edges connect pairs of boxes that are adjacent, i.e., that share a $(d-1)$-face.
- A Union-Find data structure for connected components of G. After each $\texttt{Split}(B)$, we update G and insert new `FREE` boxes into the Union-Find data structure and perform unions of new pairs of adjacent `FREE` boxes.

Let $Box_{\mathcal{T}}(\alpha)$ denote the leaf box containing α (similarly for $Box_{\mathcal{T}}(\alpha)$). The SSS Algorithm has three WHILE-loops. The first WHILE-loop will keep splitting $Box_{\mathcal{T}}(\alpha)$ until it becomes `FREE`, or declare NO-PATH when $Box_{\mathcal{T}}(\alpha)$ has radius less than ε. The second WHILE-loop does the same for $Box_{\mathcal{T}}(\beta)$. The third WHILE-loop is the main one: it will keep splitting $Q.\texttt{GetNext}()$ until a path is detected or Q is empty. If Q is empty, it returns NO-PATH. Paths are detected when the Union-Find data structure tells us that $Box_{\mathcal{T}}(\alpha)$ and $Box_{\mathcal{T}}(\beta)$ are in the same connected component. It is then easy to construct a path. Thus we get:

SSS Framework:
Input: Configurations α, β, tolerance $\varepsilon > 0$, box $B_0 \in C_{space}$.
    Initialize a subdivision tree $\mathcal{T}$ with root B_0.
    Initialize Q, G and union-find data structure.
1. While $(Box_{\mathcal{T}}(\alpha) \neq \texttt{FREE})$
        If radius of $Box_{\mathcal{T}}(\alpha))$ is $< \varepsilon$, Return(NO-PATH)
        Else $\texttt{Split}(Box_{\mathcal{T}}(\alpha))$
2. While $(Box_{\mathcal{T}}(\beta) \neq \texttt{FREE})$
        If radius of $Box_{\mathcal{T}}(\beta))$ is $< \varepsilon$, Return(NO-PATH)
        Else $\texttt{Split}(Box_{\mathcal{T}}(\beta))$
    ▷ *MAIN LOOP:*
3. While $(Find(Box_{\mathcal{T}}(\alpha)) \neq Find(Box_{\mathcal{T}}(\beta)))$
        If $Q_{\mathcal{T}}$ is empty, Return(NO-PATH)
        $B \leftarrow \mathcal{T}.\texttt{GetNext}()$
        $\texttt{Split}(B)$
4. Generate and return a path from α to β using G.

The correctness of our algorithm does not depend on how the priority of Q is designed. See [11] for the correctness of this framework under fairly general conditions.