Path Planning for Simple Robots using Soft Subdivision Search*

Ching-Hsiang Hsu, John Paul Ryan, and Chee Yap

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
New York, NY USA
{john.ryan,chhsu,chee.yap}@nyu.edu

Abstract

The concept of ε-exact path planning is a theoretically sound alternative to the standard exact algorithms, and provides much stronger guarantees than probabilistic or sampling algorithms. It opens the way for the introduction of **soft predicates** in the context of subdivision algorithm. Taking a leaf from the great success of the **Probabilistic Road Map** (PRM) framework, we formulate an analogous framework for subdivision, called **Soft Subdivision Search** (SSS). In this video, we illustrate the SSS framework for a trio of simple planar robots: disc, triangle and 2-links. These robots have, respectively, 2, 3 and 4 degrees of freedom. Our 2-link robot can also avoid self-crossing. These algorithms operate in realtime and are relatively easy to implement.

1998 ACM Subject Classification F2.2 Geometrical problems and computations, I.2.9 Autonomous vehicles

Keywords and phrases Path Planning, Configuration Space, Soft Predicates, Resolution Exactness, Subdivision Search

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.68

1 Introduction

Path Planning is a fundamental task in robotics [4, 1]. There are three main approaches: first, we have the Exact Approach which can, in principle, solve any algebraic planning problem. But practitioners tend to implement exact algorithms using machine approximations; then it is no longer clear what the guarantees of exact algorithms mean. The book [2] shows how path planning (Section 9.3), among other exact algorithms of computational geometry, may be implemented exactly (Section 1.3). Exact algorithms are impractical except for the simplest cases. Another approach is based on sampling: we refer to **Probabilistic Road Map** (PRM) [3] as the main representative. This Sampling Approach has dominated the field in the last twenty years, but its central problem is inability to halting (couched as "narrow passage problem"). Finally, we have the Subdivision Approach. This is the oldest among the three approaches, and remains popular with practitioners: our work falls under this category. In [6, 7, 8], we introduced the concept of **resolution-exactness** as the theoretical foundation for path planning that side-steps exact computation. This opens the way for the introduction of **soft predicates**, replacing the usual predicates that control the logic of all geometric algorithms (e.g., [2]).

* This work was partially supported by NSF Grant #CCF-1423228.

Subdivision methods share with Sampling methods many advantages over Exact methods. They both lead to algorithms that are easy to implement and to modify. Implementability is highly valued in a practical area like robotics. Modifiability is also important in practice because we typically deploy such algorithms in systems where non-algorithmic considerations must be accounted for. Both methods allow the possibility of discovering paths *before* the entire free configuration space has been fully explored.

This video is a demonstration of these properties. Specifically:

(a) Our algorithms are resolution-exact: in particular, it will halt on all inputs. When it returns `NO-PATH`, it guarantees there is no path of a certain specified clearance.
(b) Our algorithms are real-time: there is no pre-processing of the inputs, nor any implicit parameter selections (unlike PRM which need some additional parameters).
(c) The flexibility of our framework is seen in its ability to support a variety of global search search strategies. We implemented these strategies: random, Breadth First Search (BFS), Greedy Best First (GBF), "Voronoi heuristic".
(d) The same framework supports three distinct robots (disc, triangle, 2-link). Moreover, each of these robots is parameterizable: we can freely change the radius r_0 of the disc, the lengths (a, b, c) of the triangle, the lengths (ℓ_1, ℓ_2) of the two links.
(e) The performance of our algorithms is adaptive (not controlled by the worst case complexity). It can easily handle arbitrarily complex environments, e.g., environments involving hundreds of triangles.

Limitations. Our current software is implemented using standard machine double precision. It is not hard to extend our software to allow arbitrary precision by incorporating any standard big-float number package (this is especially easy within our Core Library [9]). It is well-known that such a step would take a performance hit. Thus our experimental claims are all based on machine precision computation, within the "normal limits" of experimental validation. What are these normal limits? Typically, the physical environment lies in a 512 square (or cube) domain. Thus our claim of "realtime performance" is within such limits. This is consistent with current practice in robotics algorithms. In a future work, we plan to carry out the error analysis to show the extent to which our results can be guaranteed with machine precision. Although the 3 robots in this video fall within our SSS framework, they remain separate pieces of software. In the future, we plan to create a common SSS framework in which all 3 robots (among others) could be supported.

2 FindPath using Soft Subdivision Search

Here we briefly summarize the SSS Framework [7]. Assume a fixed robot R_0 in physical space $\mathbb{R}^k$ (typically $k = 2, 3$) with configuration space $C_{space} = C_{space}(R_0)$. In this video review, we will see three kinds of robots R_0 as illustrated in Figure 1: their configuration spaces have 2, 3 and 4 (respectively) degrees of freedom.

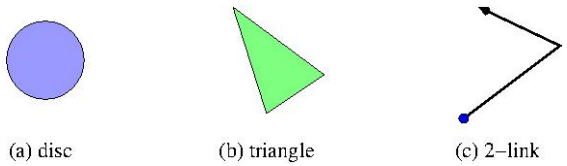

Figure 1 Three simple robots

Our demos will show two kinds of 2-link robot, depending on whether we allow or forbid the 2 links to cross each other. A configuration may be given by $(x, y, \theta_1, \theta_2) \in \mathbb{R}^2 \times \mathcal{T}^2 = C_{space}$ ($\mathcal{T}^2 = S^1 \times S^1$ is the torus). Non-crossing means that $\theta_1 = \theta_2$ is forbidden. More generally, we can forbid a band $|\theta_1 - \theta_2| \leq \delta$. This geometry is interesting since such bands do not disconnect the torus, and seems novel.

The (exact) **path planning problem** for R_0 is this: *given any polyhedral obstacle set $\Omega \subseteq \mathbb{R}^k$, and start and goal configurations $A, B \in C_{space}(R_0)$, to find an Ω-avoiding path from A to B if one exists, and return* `NO-PATH` *otherwise.* In **resolution-exact path planning**, we are given two additional input: a resolution $\varepsilon > 0$ and a region-of-interest $B_0 \subseteq C_{space}(R_0)$. There is a constant $K > 1$ independent of the inputs such that the algorithm always halts and either outputs `NO-PATH` or output an Ω-avoiding path, subject to:

(P1) If all Ω-avoiding paths in B_0 have clearance $< \varepsilon/K$, it must return `NO-PATH`.
(P2) If there exists an Ω-avoiding path in B_0 with clearance $> K\varepsilon$, it must return a path.

Note that (P1) and (P2) do not cover all possibilities: if the maximum clearance of an Ω-avoiding path is in the gap $(\varepsilon/K, K\varepsilon)$, our algorithm's output need not be deterministic.

Fix an obstacle set $\Omega \subseteq \mathbb{R}^k$. We define the **clearance** of a configuration $\gamma \in C_{space}$ to be the separation between robot in the "pose" γ and Ω. If the clearance is positive, we say γ is **free**. The **free space** $C_{free} = C_{free}(R_0, \Omega)$ comprises all such free configurations. We say γ is **semi-free** if it is on the boundary of C_{free}.

The search for path is restricted to B_0, which will be recursively split into subboxes $B \subseteq B_0$. We focus on box predicates $\widetilde{C} : B \mapsto \{\texttt{FREE}, \texttt{STUCK}, \texttt{MIXED}\}$. The box predicate $\widetilde{C}$ is **conservative** if $\begin{cases} \widetilde{C}(B) = \texttt{FREE} & \text{implies} \quad B \subseteq C_{free} \\ \widetilde{C}(B) = \texttt{STUCK} & \text{implies} \quad B \cap C_{free} = \emptyset \end{cases}$

A maximally conservative predicate is trivial — it always outputs `MIXED`. A minimally conservative predicate is called **exact** — it outputs `FREE` or `STUCK` whenever possible. We say $\widetilde{C}$ is **convergent** if for any monotone sequence $\{B_i : i \geq 0\}$ that converges to a point $p = \lim_{i\to\infty} B_i$, we have $\widetilde{C}(B_i) = C(p)$ for i large enough. Here, $C(p)$ denotes the exact predicate with $C(p) = \texttt{MIXED}$ iff p is semifree. We say $\widetilde{C}$ is a **soft predicate** if it is conservative and convergent. The design of soft predicates is of considerable interest, and illustrated in [6, 8]; there, we further show how soft predicates with the additional property of **effectivity** leads to resolution-exact path planners, called **SSS planners**. Briefly, the main loop of SSS planners is controlled by a priority queue Q containing `MIXED` boxes. While Q is non-empty, we remove a box B from Q, subdivide B into children, classify them and put the `MIXED` boxes back into Q. We maintain the connected components of all the free boxes using the well-known **Union-Find** data structure; two boxes are unioned if they are adjacent. More precisely, the SSS planner has three plug-in subroutines:

(S1) A soft predicate $\widetilde{C}(B)$.
(S2) A global search strategy to determine the priority of boxes in Q: this is encoded in the Q.`getNext`() method which returns a box from Q. Correctness of SSS does not depend on the global strategy.
(S3) A subroutine `Expand`(B) that could fail: first, remove B from Q. If B has width $< \varepsilon$, then `Expand`(B) fails. Otherwise, we subdivide B into a set `S`$split(B)$ with two or more subboxes. For each $B' \in$ `S`$split(B)$, we compute $\widetilde{C}(B')$. If $\widetilde{C}(B') = \texttt{FREE}$, we add B' into a Union-Find structure, and union B' with each adjacent box B'' already in the structure. If $\widetilde{C}(B') = \texttt{MIXED}$, we put B' into Q.

Finally, we put together these 3 subroutines. Let $Box(\alpha)$ denote a smallest subdivision box that contains α. We can keep track of $Box(\alpha)$ and $Box(\beta)$.

SSS FindPath:
Input: Configurations α, β, tolerance $\varepsilon > 0$, box $B_0 \subseteq C_{space}$.
Output: Either path from α to β, or `NO-PATH`.
Initialize $Q = \{B_0\}$.
1. While ($Box(\alpha) \neq$ `FREE`)
   If (`Expand`($Box(\alpha)$) fails), Return(`NO-PATH`).
2. While ($Box(\beta) \neq$ `FREE`)
   If (`Expand`($Box(\beta)$) fails), Return(`NO-PATH`).
3. While ($Find(Box(\alpha)) \neq Find(Box(\beta))$)
   If Q is empty, Return(`NO-PATH`)
   `Expand`($Q.getNext()$)
4. Compute a channel P from $Box(\alpha)$ to $Box(\beta)$.
   Return a path in this channel.

The **channel** in Step 4 is a sequence $(B_1, \ldots, B_m)$ of free boxes where B_i, B_{i+1} are adjacent. The expansion technique for our 2-link robot is non-standard in order to achieve our performance (see [5]).

Acknowledgments The original software for the disc and triangle robot was implemented as part of Cong Wang's PhD thesis and reported in [6]. The 2-link robot was implemented in Luo's Masters thesis and reported in [5]. The original graphics was written in OpenGL 2.1 using GLUT/GLUI libraries. In the summer of 2015, Bryant Curto and John Ryan (supported by a departmental Undergraduate Summer Research Fellowship) re-implemented the disc and 2-link software in the Qt IDE (OpenGL 4.x) resulting in much faster graphics. Ching-Hsiang Hsu re-implemented the triangle software in Qt.

References

1 H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion*. MIT Press, Boston, 2005.
2 Dan Halperin, Efi Fogel, and Ron Wein. *CGAL Arrangements and Their Applications*. Springer-Verlag, Berlin and Heidelberg, 2012.
3 Lydia Kavraki, P. Švestka, Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.
4 Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.
5 Zhongdi Luo, Yi-Jen Chiang, Jyh-Ming Lien, and Chee Yap. Resolution exact algorithms for link robots. In *Proc. 11th Intl. Workshop on Algorithmic Foundations of Robotics (WAFR)*, vol. 107 of *Springer Tracts in Advanced Robotics (STAR)*, pp. 353–370, 2015.
6 Cong Wang, Yi-Jen Chiang, and Chee Yap. On Soft Predicates in Subdivision Motion Planning. *Comput. Geometry: Theory and Appl.*, 48(8):589–605, September 2015.
7 Chee K. Yap. Soft Subdivision Search in Motion Planning. In A. Aladren et al., editor, *Proceedings, 1st Workshop on Robotics Challenge and Vision (RCV 2013)*, 2013. **Best Paper Award**. In arXiv:1402.3213.
8 Chee K. Yap. Soft Subdivision Search and Motion Planning, II: Axiomatics. In *Frontiers in Algorithmics*, volume 9130 of *Lecture Notes in Comp.Sci.*, pages 7–22. Springer, 2015. Invited. 9th FAW. Guilin, China. Aug 3-5, 2015.
9 Jihun Yu, Chee Yap, Zilin Du, Sylvain Pion, and Herve Bronnimann. Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In *Proc. 3rd ICMS*, pages 121–141. Springer, 2010. LNCS No. 6327.