Computing push plans for disk-shaped robots

Computing Push Plans for Disk-Shaped Robots

Mark de Berg* Dirk H.P. Gerrits*

Abstract

Suppose we want to move a passive object along a given path, among obstacles in the plane, by pushing it with an active robot. We present two algorithms to compute a push plan for the case that the obstacles are non-intersecting line segments, and the object and robot are disks. The first algorithm assumes that the robot must maintain contact with the object at all times, and produces a shortest path. There are also situations, however, where the robot has no choice but to let go of the object occasionally. Our second algorithm handles such cases, but no longer guarantees that the produced path is the shortest possible.

1 Introduction

A fundamental problem in robotics is *path planning* [9], in which a robot has to find ways to navigate through its environment from its initial configuration to a certain destination configuration, without bumping into obstacles. Many variants of this problem have been studied, involving widely differing models for the environment, and for the robot and its movement. In *manipulation path planning* [7] the robot's goal is to make a passive object, rather than the robot itself, reach a certain destination. Several different kinds of manipulation have been studied, including grasping [7], squeezing [5], rolling [2], and even throwing [8].

The manipulation path planning problem studied here involves *pushing* [7]. In particular, we want a disk-shaped robot to push a disk-shaped object to a given destination in the plane among polygonal obstacles. Nieuwenhuisen et al. [9, 11, 12] developed a probabilistically complete algorithm for this based on the *Rapidly-exploring Random Trees* path-planning algorithm [6]. This builds a tree of reachable positions by repeatedly generating object paths and trying whether the pusher can make the object follow such a path. Thus a subroutine is needed to push the object along a given path.

Problem statement. In the plane, let P be a disk-shaped robot (of radius r_p) called the *pusher*, let O be a disk-shaped *object* (of radius $r_o > r_p$), and let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a set of non-intersecting line segments called the *obstacles*. We're given a collision-free path τ for O consisting of k constant-complexity curves $\tau_1, \ldots, \tau_k$ called the *path sections*. We then want to compute a collision-free path σ for P such that P pushes O along τ when P moves along σ. We allow P and O to slide along obstacles, which is called a *compliant* motion. The computed path σ will be called a *push plan*. We distinguish two kinds of push plans: *contact-preserving push plans* in which the pusher maintains contact with the object at all times, and *unrestricted push plans* in which the pusher can occasionally let go of the object.

Related work. Along with the algorithm described above, Nieuwenhuisen et al. [9, 10] also developed a subroutine needed by the algorithm that solves the problem just described. They assume the object path consists of line segments and circular arcs only, and after preprocessing the n obstacles in $O(n^2 \log n)$ time into an $O(n^2)$-space data structure, they can compute a contact-preserving push plan in $O(kn \log n)$ time. It is not guaranteed that the constructed push plan is optimal in any way.

Agarwal et al. [1] considered the problem where only the final destination of the object is given, and not its path τ. For this they give an algorithm for finding a contact-preserving push plan for a point-size pusher and a unit-disk object. The algorithm discretizes the problem in two ways: the angle at which the pusher can push is constrained to $1/\varepsilon$ different values, and the combined boundary of the obstacles is sampled at m locations to give potential intermediate positions for the object. The algorithm then runs in $O((1/\varepsilon)m(m+n)\log n)$ time, but is only guaranteed to find a solution if $1/\varepsilon$ and m are large enough. The algorithm assumes the pusher can get to any position around the object at all times, which is true for their point-sized pusher but not for our disk-shaped pusher: there may be obstacles in the way.

Our results. We present a new algorithm for computing contact-preserving push plans for a given object path. It matches the $O(kn \log n)$ time needed by Nieuwenhuisen's approach, while handling path sections other than line segments and circular arcs, and without needing a $O(n^2 \log n)$-time and $O(n^2)$-space preprocessing step. We also present the first algorithms to compute *shortest* contact-preserving push plans, in $O(k^2 n^2 \log(kn))$ time and $O(k^2 n^2)$ space, and to to compute unrestricted push plans, in $O(kn \log(kn) + kn^2 \log n)$ time and $O(kn^2)$ space.

*Dept. of Computer Science, TU Eindhoven, the Netherlands, mdberg@win.tue.nl, dirk@dirkgerrits.com

If we assume the obstacles aren't too densely packed, these time and space bounds can be greatly improved. Both Nieuwenhuisen's and our approach for computing contact-preserving push plans then take $O((k+n)\log(k+n))$ time and $O(k+n)$ space, but the former still needs its expensive preprocessing step. For shortest contact-preserving push plans our approach takes $O((k+n)\log(k+n)+k^2\log k)$ time and $O(k+n)$ space, and for unrestricted push plans it takes $O((k+n)\log(k+n)+kn)$ time and $O(kn)$ space. For lack of space we only present the high-obstacle-density results here, and omit some proofs. Details can be found in Gerrits's MSc thesis [4].

2 Preliminaries

In the problem statement we've been deliberately vague about what pushing the object along a path entails. Before discussing our algorithms we fill in some of the details.

The push range. As mentioned before, *compliant motions* are motions where the object slides along an obstacle. Such motions are more robust in the presence of sensor inaccuracies, because the obstacle will act as a guide for the object. More importantly, this allows the pusher to achieve the same motion for the object from a continuous range of different pushing positions, called the *push range*. The pusher can then swerve around the object to avoid obstacles while still pushing the object in the desired direction. (See Figure 1(a).) With a non-compliant motion, the push range is a single pushing position depending only on the desired direction of motion for the object. If any obstacles are in the way, there simply exists no push plan for that object motion. (See Figure 1(b).)

(a) (b)

Figure 1: The push range for (a) a compliant motion, and (b) a non-compliant motion.

Friction. The exact size of the push range for compliant motions depends on the friction characteristics of the two disks and the obstacles. Nieuwenhuisen [9] describes how to compute the push range, given the friction coefficients between the disks and between the object and obstacles. Friction also affects how pushing works for non-compliant motions. Agarwal et al. [1] studied the motion of the object resulting from pushing in a straight line under simple friction assumptions.

We abstract away from compliance and friction by assuming that we can compute the push range for any position of the object along its path. We assume (as do Nieuwenhuisen and Agarwal et al.) that pushing is *quasi-static* [13], i.e. when pushing stops, the object also stops instantly. (This will never be the case in reality, but it can be closely approximated by pushing slowly, or having very high friction between the disks and the floor.)

Assumptions about the input path. We assume the given path $\tau : [0,1] \to \mathbb{R}^2$ for the object does not take it through any of the obstacles. Furthermore, we assume that τ is made up of k constant-complexity curves $\tau_1, \ldots, \tau_k$. We further assume that the path sections are "well-behaved" (in a technical sense explained in Gerrits's MSc thesis [4]). The line segments and circular arcs used as path sections by Nieuwenhuisen are all well-behaved.

3 Pushing while maintaining contact

A general-purpose technique for path-planning is to translate the problem from the *work space* into the *configuration space*. The work space is the environment in which the robot has to find a path, and a *configuration* is one specific placement of the robot in this space. Each point in the configuration space corresponds to a configuration in the work space. Some configurations are invalid because the robot would intersect an obstacle and these form the *forbidden (configuration) space*. The remainder is the *free (configuration) space*, and a path through it translates back to a solution to the original path-planning problem in the work space.

To apply this technique to our problem, we first discuss what our configuration space looks like, then how to compute it and how to find a path through it.

Shape of the configuration space. In our case, a configuration is a placement of both the pusher and the object in the work space. Since the object is restricted to the path τ, and we assume that the pusher and object can maintain contact at all times (for now; we will lift this restriction in Section 4), the configuration space is two-dimensional. The point $(s,\theta) \in [0,1] \times \mathcal{S}^1$ in the configuration space will represent the configuration with the object's center at $\tau(s)$ and with θ being the *pushing angle*: the angle that the line from the pusher's center to the object's center makes with the positive x-axis. (Note that the configuration space is cylindrical, but for clarity we will depict it "flattened" as a rectangle.)

We assume that path τ does not take the object through any obstacles, so a configuration can be invalid for only two reasons: either the pusher intersects an obstacle, or the pusher is outside of the push range.

We therefore consider the forbidden space to be the union of two kinds of shapes. A *configuration-space obstacle* $\mathcal{C}_\gamma$ consists of the configurations where the pusher intersects obstacle γ. A *forbidden push range* FPR_i consists of the configurations where the object is on the interior of path section τ_i and the pusher is outside the push range. By $\mathcal{C}_{\gamma,i}$ we'll mean the restriction of $\mathcal{C}_\gamma$ to configurations with the object on path section τ_i. The forbidden space is then the union of these $k(n+1)$ shapes (n obstacles and 1 forbidden push range per path section). An example is shown in Figure 2.

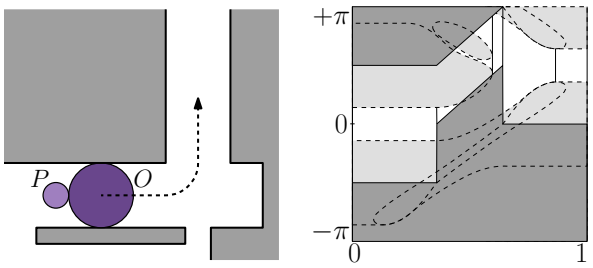

Figure 2: An example work space and its configuration space. The s-axis is horizontal, the θ-axis is vertical. Configuration-space obstacles are drawn dashed in light gray, the forbidden push range is drawn in dark gray.

Theorem 1 *The configuration space has complexity $O(kn)$, i.e. the boundary of the forbidden space consists of $O(kn)$ vertices and constant-complexity curves between them.*

Proof. Since a path section τ_i has constant complexity, so do FPR_i and $\mathcal{C}_{\gamma,i}$ for all $\gamma \in \Gamma$. We will prove that $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. It then follows that $\mathrm{FPR}_i \cup \bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$, the forbidden space for one path section, also has complexity $O(n)$, yielding $O(kn)$ in total.

The boundary of $\mathcal{C}_\gamma$ corresponds to configurations where the pusher is compliant with γ. Such pusher positions all lie at distance r_p from γ and thus form a "capsule." A point of intersection of $\mathcal{C}_{\gamma_1,i}$ and $\mathcal{C}_{\gamma_2,i}$ corresponds to a configuration where the pusher is compliant with both γ_1 and γ_2, and that pusher position must thus be the intersection of the corresponding capsules. These capsules form a collection of *pseudodisks* [3, Chapter 13], and therefore have a union complexity of $O(n)$. Thus there can only be $O(n)$ positions where the pusher would be compliant with more than one obstacle. Each of these pusher positions could show up in the configuration space more than once, since path τ_i could take the object past this point multiple times. However, this cannot happen more than $O(1)$ times, since τ_i is well-behaved. Thus $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. □

Computing the configuration space. To compute the configuration space in a form that allows us to easily compute a push plan we perform the following steps:

1. Compute $\mathcal{C}_{\gamma,i}$ for all $\gamma \in \Gamma$, and all $\tau_i \in \tau$.
2. Compute FPR_i for all $\tau_i \in \tau$.
3. Take the union of these shapes to get the forbidden space.
4. Divide the free space into cells by a vertical decomposition.
5. Create the *cell graph* by directing an edge from cell c_1 to c_2 iff c_1's right boundary touches c_2's left boundary.

The running time of this approach is expressed by the following theorem:

Theorem 2 *The configuration space can be computed in $O(kn\log^2 n)$ time worst-case, or $O(kn\log n)$ expected time, both using $O(kn)$ space.*

Computing a shortest contact-preserving push plan. Not every path through the free space actually yields a push plan. If a path through the configuration space is not s-monotone then the pusher would have to pull the object on occasion. To prevent this we remove from the cell graph all cells that are not reachable from the starting configuration by a valid contact-preserving push plan. It's then fairly simple to find an arbitrary s-monotone path in linear time, resulting in a contact-preserving push plan in the time and space bounds of Theorem 2.

It is tempting to instead compute a Euclidean shortest path through the reachable cells. The cells themselves are s-monotone, so a shortest path through the remaining cells must yield a push plan. Unfortunately, a shortest path in configuration space does not necessarily minimize the pusher's movement in the work space. We can circumvent this problem by performing our computations in the work space. Each cell of the free space decomposition corresponds to a contiguous subset of the valid configurations. The pusher positions of these configurations also form a contiguous region in the work space. We could call this region the corresponding *work-space cell*. All work-space cells together form the region that P may move in to accomplish O's desired motion. Computing a shortest path [14] through this region then yields a shortest contact-preserving push plan.

Theorem 3 *Given the configuration space a shortest contact-preserving push plan can be computed in $O(k^2n^2\log(kn))$ time and $O(k^2n^2)$ space.*

4 Pushing and releasing

Until now we've assumed the pusher can maintain contact with the object at all times. However, the situation that was depicted in Figure 2 does not admit such contact-preserving push plans. (In fact, we've proven [4] that this is the case for *any* object path with the same start and end point.) It does admit an unrestricted push plan, as can be seen in Figure 3.

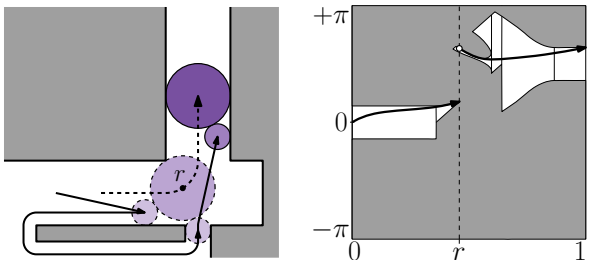

Figure 3: An unrestricted push plan for the example of Figure 2, doing one release at position r.

Canonical releasing positions. Whenever the push range is split into multiple contiguous ranges by obstacles, it may make sense for P to let go of O and try to reach one of these other positions. In the configuration space this situation corresponds to a vertical line intersecting multiple cells. In general there are infinitely many such *potential releasing positions*, thus it's infeasible to try them all. Instead we consider only vertical lines that go through a vertex of a cell or configuration-space obstacle. We call the resulting set of $O(kn)$ positions the *canonical releasing positions*. We've proved that if an unrestricted push plan exists, then there is also an unrestricted push plan where P only releases O at canonical releasing positions [4].

Computing an unrestricted push plan. Restricting ourselves to canonical releasing positions, we cannot guarantee a shortest push plan anymore. Thus we can abandon the work-space-cell approach and instead proceed as follows:

1. Compute a road map [3, Chapter 13] $\mathcal{S}$ for P among the obstacles.
2. At each canonical releasing point r, determine the set of cells intersected by the vertical line through r. Add O as an extra obstacle in $\mathcal{S}$ to get $\mathcal{S}'$, and determine for each intersected cell in which component of $\mathcal{S}'$ its pusher positions lie. Add edges in the cell graph between cells sharing a component of $\mathcal{S}'$.
3. Compute a path through this extended cell graph.
4. Convert the path into a push plan. For edges of the original cell graph this is straightforward, for the extra edges use $\mathcal{S}'$ to find a path for P.

The running time of this approach is expressed by the following theorem:

Theorem 4 *Given the configuration space an unrestricted push plan can be computed in $O(kn\log(kn)+kn^2\log n)$ time and $O(kn^2)$ space.*

References

[1] P. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proc. IEEE Int. Conf. Robotics & Automation*, volume 4, pages 3124–3129, 1997.

[2] H. Arai and O. Khatib. Experiments with dynamic skills. In *Proc. Japan-USA Symp. Flexible Automation*, pages 81–84, 1994.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[4] D. Gerrits. Designing push plans for disk-shaped robots. Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2008.

[5] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2–4):210–225, 1993.

[6] S. LaValle and J. Kuffner. Rapidly-exploring random trees. In B. Donald, K. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[7] M. Mason. *Mechanics of Robotic Manipulation*. Intelligent Robots & Autonomous Agents. MIT Press, 2001.

[8] M. Mason and K. Lynch. Dynamic manipulation. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, pages 152–159, 1993.

[9] D. Nieuwenhuisen. *Path Planning in Changeable Environments*. PhD thesis, Universiteit Utrecht, The Netherlands, 2007.

[10] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Path planning for pushing a disk using compliance. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, pages 4061–4067, 2005.

[11] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Pushing using compliance. In *Proc. IEEE Int. Conf. Robotics & Automation*, pages 2010–2016, 2006.

[12] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Pushing a disk using compliance. *IEEE Transactions on Robotics*, 23(3):431–442, 2007.

[13] M. Peshkin and A. Sanderson. Minimization of energy in quasi-static manipulation. *IEEE Transactions on Robotics & Automation*, 5(1):53–60, 1989.

[14] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proc. 12th ACM Symp. Comput. Geom.*, pages 248–257, 1995.