International Journal of Computational Geometry
& Applications
Vol. 23, No. 1 (2013) 29–48

DOI: 10.1142/S0218195913500027

World Scientific
www.worldscientific.com

COMPUTING PUSH PLANS FOR DISK-SHAPED ROBOTS

MARK DE BERG* and DIRK H. P. GERRITS†
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
Den Dolech 2, 5600 MB Eindhoven, The Netherlands
**mdberg@win.tue.nl*
†*dirk@dirkgerrits.com*

Received 9 March 2011
Revised 17 October 2012
Communicated by Nancy Amato

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 object and robot are disks and the obstacles are non-intersecting line segments. (When only the object's destination and not its full path is given these algorithms can still be used as subroutines in a larger algorithm to compute such a path.) 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.

Keywords: Path planning; pushing.

1. Introduction

A fundamental problem in robotics is *path planning*,[1] 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*[2] 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,[2] squeezing,[3] rolling,[4] and even throwing.[5]

The manipulation path-planning problem studied here involves *pushing*.[2] 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.*[6] developed a probabilistically-complete algorithm for this based on the *Rapidly-exploring*

Random Trees path-planning algorithm.[7] This algorithm 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.

1.1. *Problem statement*

Let P be a disk-shaped *pusher* robot of radius r_p in the plane, 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*. The pusher P is holonomic: it can move freely in two dimensions. For the object O we are given a collision-free path τ to follow, consisting of k *convex*, *constant-complexity* curves $\tau_1, \ldots, \tau_k$ called the *path sections*. A curve from a to b is called convex if the region bounded by the curve together with the line segment $\overline{ab}$ is convex. By a constant-complexity curve we mean that such a curve takes $O(1)$ space to represent, and that we can perform several basic operations on them in $O(1)$ time: computing tangents through a point for a curve, and computing bi-tangents and points of intersection between two curves.

Given τ in this manner, we 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.

1.2. *Related work*

Nieuwenhuisen *et al.*[6] present an algorithm for this problem that assumes the object path consists only of line segments and circular arcs. 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. If one assumes low obstacle density,[8] then the latter bound can be improved to $O((k+n)\log(k+n))$. In neither case does their algorithm guarantee that the constructed push plan is optimal in any way.

Agarwal *et al.*[9] consider the problem where only the final destination of the object is given directly, without such a subroutine for a given object 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-size pusher but not for our disk-shaped pusher: there may be obstacles in the way.

1.3. *Our results*

We present a new approach to compute push plans for a disk-shaped robot pushing a disk-shaped object along a given path. It improves on the method of Nieuwenhuisen *et al.* in several ways. First, our method can compute *shortest* contact-preserving push plans, minimizing the distance traveled by the pusher. Second, it can be generalized to computing *unrestricted* push plans. Finally, our approach can deal with more general paths than the method of Nieuwenhuisen *et al.* Table 1 summarizes our results and those of Nieuwenhuisen *et al.*, both for high and low obstacle density. Note that our algorithms are not only more powerful, they also have better running times; in particular, we need neither $O(n^2 \log n)$ time nor $O(n^2)$ space for preprocessing.

Table 1. A comparison of the asymptotic running times of Nieuwenhuisen's approach and ours for computing contact-preserving (CPPP) and unrestricted push plans (UPP).

	High obstacle density		Low obstacle density	
	Nieuwenhuisen	Our method	Nieuwenhuisen	Our method
Preprocessing	$n^2 \log n$	$n \log n$ $^{(*)}$	$n^2 \log n$	$n \log n$ $^{(*)}$
Any CPPP	$kn \log n$	$kn \log n$ $^{(*)}$	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
A shortest CPPP	—	$kn \log n$ $^{(*)}$	—	$kn \log n$ $^{(*)}$
Any UPP	—	$kn \log k + kn^2 \log n$	—	$(k+n)\log(k+n) + kn$

Note: Entries marked $^{(*)}$ are expected times. For worst-case times, replace $\log n$ by $\log^2 n$.

2. The Configuration Space

A general-purpose technique for path planning is to translate the problem from the *work space* into the *configuration space*.[1] The work space is the environment in which the robot has to find a path. A *configuration* is one specific placement of the robot in this space, specified by f parameters, where f is the number of degrees of freedom of the robot. Each point in the f-dimensional 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 represents a solution to the original path-planning problem in the work space.

To apply this technique to our problem, we first clarify a few details left out from the problem statement, and then discuss what our configuration space looks like and how to compute it. Finding paths through the configuration space, and mapping these back to push plans, is discussed in Secs. 3 and 4.

2.1. *Preliminaries*

We assume (as do Nieuwenhuisen and Agarwal *et al.*) that pushing is *quasi-static*.[11] That is, when pushing stops the object also stops instantly. This is realistic if

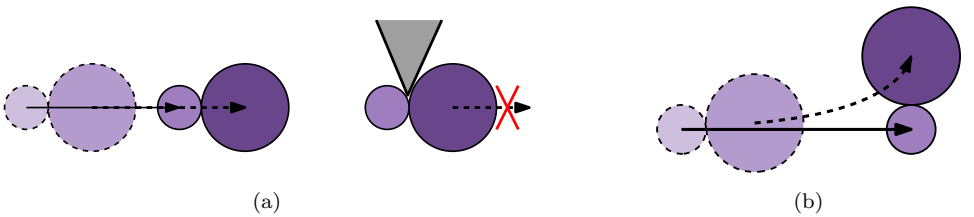


Fig. 1. (a) A straight-line non-compliant path section, and (b) a hockey-stick non-compliant path section. In this figure and all that follow, the pusher P is depicted as the smaller, lighter disk and the object O as the larger, darker disk. Obstacles are shown as fat, black lines and the areas they bound are filled with gray. Motion is depicted by arrows; the boundaries of P and O at the current position are solid, and are dashed at past or future positions.

pushing is done very slowly, or if there is a large amount of friction between the disks and the floor. To push the object along a straight-line path in the absence of obstacles, the ray from the pusher's center to the object's center will have to line up *exactly* with the desired direction of motion, as in Fig. 1(a). When pushing even slightly off-center, the object's motion will instead be curved. If we assume the point of contact between the two disks never "slips" during a push, the result is a so-called *hockey-stick curve*,[9] as seen in Fig. 1(b). Given such a straight-line or hockey-stick motion for the object, the pusher motion that will accomplish it is fixed.

In contrast to such *non-compliant motions*, a *compliant motion* uses obstacles as a guide for the object, as in Fig. 2. On the left the object is pushed straight along the length of an obstacle, in the middle the object is pushed in a circular arc around an obstacle endpoint. Compliant motions are more robust in the presence of sensor inaccuracies, and allow the pusher to avoid obstacles more easily, as the same object motion can be achieved by a whole range of pusher motions. At any point in time, the set of allowed positions for the pusher's center from which pushing would result in the desired object motion form a circular arc called the *push range*. The size of this arc can be computed from the friction coefficients between the pusher and the object and between the object and the obstacle.[10]

We assume that the given object path $\tau : [0, 1] \rightarrow \mathbb{R}^2$ consists of k constant-complexity pieces $\tau_1, \ldots, \tau_k$ connected end-to-end. We call each piece a *path section*. A path section represents one of the four aforementioned motions: a straight-line or hockey-stick non-compliant motion, or a straight-line or circular compliant motion. To avoid four-way case distinctions in the rest of this paper, we introduce an abstract *well-behaved* path section which generalizes these four. As an added benefit, this makes our algorithm more general: we can easily handle other types of path sections, as long as they are well-behaved. For example, one may wish to incorporate non-compliant motions for which the pusher has to follow a non-straight path, or in which the contact between the two disks *does* slip during the push. We call a path section τ_i well-behaved if it is a convex, constant-complexity curve and satisfies the

following conditions (see Figs. 2 and 3):

(W1) We can compute the push range $\mathrm{pr}_i(s)$ for any object position $\tau_i(s)$ along τ_i in $O(1)$ time. Furthermore, this push range is such that the ray from O's center in the direction of τ_i always forms an angle of more than 90° with a ray from O's center to a pushing position. (This is natural, since otherwise the pusher would pull the object rather than push it.)

(W2) Let $\mathcal{A} = \bigcup_{s\in[0,1]} \mathrm{pr}_i(s)$ be the area swept out by the push range as the object moves along τ_i. Then $\mathcal{A}$ does not intersect itself, and $\mathcal{A}$ is bounded by four convex, constant-complexity curves (the push ranges at either end of τ_i, and the paths traced out by the two end points of the push range) which can be computed in $O(1)$ time.

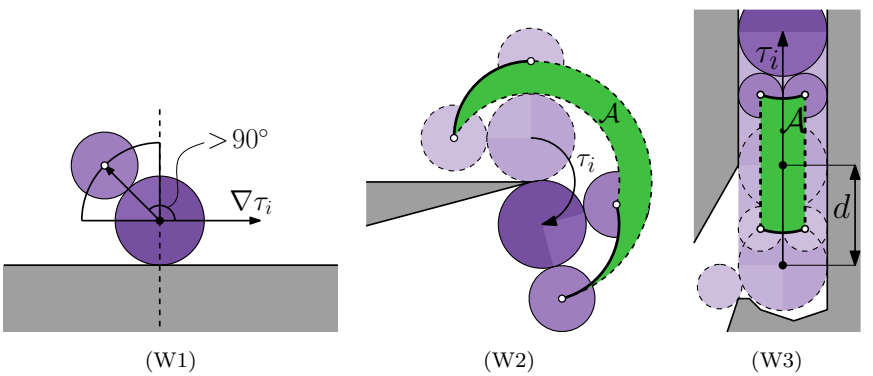


Fig. 2. The three properties satisfied by well-behaved path sections. The push range is shown at a few locations with a sector on the object along with a circular arc outside the object giving the valid locations for the pusher's center.

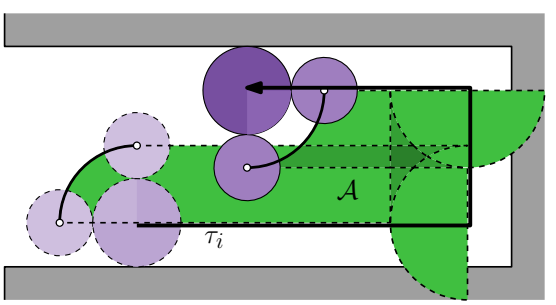


Fig. 3. A path section where well-behavedness property (W2) is *not* satisfied (the area swept out by the push range self-intersects). The path section *can* be broken up into three well-behaved path sections.

(W3) For compliant sections there is a constant $d = O(r_o)$ such that, after the object has moved a distance d along the path section, the push range becomes such that the pusher can remain in the sweep area of the object for the rest of the section. Furthermore, the smaller push range that keeps the pusher in the object's sweep area satisfies (W2).

For correctness of our algorithms, (W1) and (W2) suffice, but (W3) allows us to derive better running times in case the obstacles are not too densely packed, as will be discussed in Sec. 2.4. Condition (W3) basically states that for long path sections we need only maneuver around obstacles during the beginning of the path section. In compliant sections any later obstacles can be avoided by "hiding" behind the object, and in non-compliant sections obstacles cannot be avoided at all. Note that circular compliant sections satisfy (W3) trivially as their total length is $O(r_o)$, and straight-line compliant path sections can be shown to satisfy (W3) as well,[10] as seen informally on the right in Fig. 2.

It is then fairly easy to see that all three criteria are satisfied by straight-line and circular compliant path section, as well as by straight-line non-compliant path sections. As these are the path sections handled by the algorithm of Nieuwenhuisen *et al.*,[6] our algorithm is at least as general as theirs. A hockey-stick curve (whose equation is given in Appendix A) is not strictly of constant complexity by our definition, but it can be closely approximated to any desired degree by a sequence of quadratic Bézier curves. Thus hockey-stick non-compliant path sections can also be seen as well-behaved, and we can handle them uniformly with other types of path sections. In Nieuwenhuisen's approach they are handled outside of the main algorithm by ad hoc numerical methods.[6]

2.2. *Shape of the configuration space*

A configuration in our problem is a placement of both the pusher and the object in the work space. Recall that the object is restricted to move along a given path τ, and assume for now that the pusher and object maintain contact at all times (this latter restriction will be lifted in Sec. 4). Under these conditions, the configuration space is 2-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 ray 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

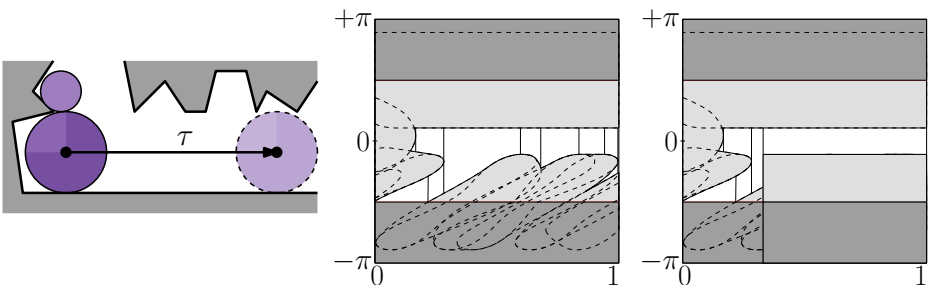


Fig. 4. (a) An example work space, (b) its configuration space, and (c) its reduced configuration space (defined below). The s-axis is horizontal, the θ-axis is vertical. Configuration-space obstacles are drawn dashed in light gray, the forbidden push range is drawn in dark gray.

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 $k(n+1)$ shapes: n obstacles and one forbidden push range for each of the k path sections. Figures 4(a) and 4(b) show an example for one path section.

Theorem 1. *The configuration space for each path section has complexity $O(n)$ (that is, the boundary of the forbidden space consists of $O(n)$ vertices and constant-complexity curves between them), and thus the total configuration space has complexity $O(kn)$.*

Proof. Since a path section τ_i has constant complexity, so do FPR_i and $\mathcal{C}_{\gamma,i}$ for all obstacles $\gamma \in \Gamma$. 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 the boundary of the "capsule" $\gamma \oplus D(r_p)$. Here "$\oplus$" denotes the Minkowski sum ($A \oplus B = \{\, x + y \mid x \in A,\, y \in B \,\}$), and $D(r)$ denotes an origin-centered disk of radius r. 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 an intersection of the corresponding capsules. These capsules form a collection of *pseudodisks*,[12] 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 a convex, constant-complexity curve. Thus $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. □

2.3. *Computing the configuration space*

To compute the configuration space in a form that allows us to easily compute a push plan, we do the following:

(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* of the decomposition. Nodes in this graph correspond to cells and there is a directed edge from cell c_1 to c_2 if and only if 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.*

Proof. For each of the k path sections, Steps 1 and 2 can be performed in $O(n)$ time as each of these $n+1$ shapes has $O(1)$ complexity. The curves bounding these shapes have complex equations not amenable to exact intersection computations needed for Step 3. However we can work with work-space analogues of these shapes to compute their vertices and combinatorial structure. For example, the points of vertical tangency of $\mathcal{C}_{\gamma,i}$ correspond to intersections of τ_i with the capsule $\gamma \oplus D(2r_o + r_p)$, and the intersection of $\mathcal{C}_{\gamma,i}$ with the vertical line $s = s'$ corresponds to the intersection of $\gamma \oplus D(r_p)$ with a circle of radius $2r_o + r_p$ centered at $\tau_i(s')$. Thus, all basic operations required (such as deciding whether one point of vertical tangency lies to the left or to the right of another) can be performed in $O(1)$ time.

With this "trick", Step 3 can then be performed by a deterministic algorithm by Kedem *et al.*[12] that uses $O(n \log^2 n)$ time, or a randomized incremental algorithm by Miller and Sharir[13] that uses $O(n \log n)$ expected time, both using $O(n)$ space. For Steps 4 and 5 we extend vertical lines from each of the $O(n)$ vertices (including points of vertical tangency) of the forbidden space. This yields a vertical decomposition of the free space into cells. These cells can be computed and connected together with a sweep-line algorithm in $O(n \log n)$ time and $O(n)$ space. □

2.4. *Low obstacle density*

The asymptotic upper bounds derived for our algorithm so far assume nothing about how densely packed the obstacles are. An environment Γ is said to have an obstacle density of λ if λ is the smallest positive number for which any disk D intersects at most λ obstacles $\gamma \in \Gamma$ with $\mathrm{length}(\gamma) \geq \mathrm{diam}(D)$. By property (W3) of well-behaved compliant path sections there is a constant $d = O(r_o)$ such that, after the object has been pushed a distance d along a path section, the pusher can then remain in the (obstacle-free) sweep area of the object for the rest of the section.

The combined sweep area of the object and pusher for this length-d "prefix" of the path section fits in a disk of diameter $d + 2r_o + 4r_p = O(r_o)$, and can thus intersect at most $O(\lambda \cdot {r_o}^2/\delta^2)$ obstacles, where δ is the length of the shortest obstacle. Assuming constant λ and $\delta = \Omega(r_o)$, the configuration space for this prefix has constant complexity.

The complexity of the remaining "suffix" of the path section can be $\Omega(n)$, though, so the total complexity of the configuration space can still be $\Omega(kn)$. However, for this suffix we can replace the $O(n)$-complexity shape $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ by the $O(1)$-complexity shape that forces the pusher to remain in the object's sweep area. This yields the *reduced configuration space* (see Fig. 4(c)), which admits a push plan if and only if the original configuration space does, but has lower complexity and can be computed more quickly:

Theorem 3. *Assuming constant λ and $\delta = \Omega(r_o)$, the reduced configuration space has complexity $O(1)$ per path section (that is, $O(k)$ in total), and can be computed in $O((k+n)\log(k+n))$ time using $O(k+n)$ space.*

Proof. For the reduced configuration space, computing $\mathcal{C}_{\gamma,i}$ individually for each $\gamma \in \Gamma$ and $\tau_i \in \tau$ is wasteful. Most of these kn shapes will either be empty (if the distance between τ_i and γ is too great), or irrelevant (if τ_i is compliant and only comes close to γ past its length-d prefix). Let τ_i' denote the length-d prefix of τ_i if τ_i is a compliant section, and $\tau_i' = \tau_i$ otherwise. Then $\mathcal{C}_{\gamma,i}$ is relevant and non-empty if and only if the curve τ_i' intersects the capsule $\gamma \oplus D(2r_o + r_p)$. We use an algorithm due to Balaban[14] to find the I points of intersections between these k curves and n capsules in $O((k+n)\log(k+n) + I)$ time and $O(k+n)$ space. From the above discussion it follows that each compliant path section contributes only $O(1)$ to I for the reduced configuration space. If non-compliant sections contribute any intersections, then no push plan can exist and we can abort the intersection computation. Thus, we can determine the relevant obstacles for all path sections in $O((k+n)\log(k+n))$ time. Since each section only has $O(1)$ relevant obstacles, the remaining steps to compute the reduced configuration space can be done in $O(1)$ time per path section. □

3. Pushing While Maintaining Contact

Not every path through the free space actually yields a push plan. The path through the configuration space needs to be *s-monotone*, that is, any "vertical line" (having a constant value for s) must intersect the path in at most one point. If a path through the configuration space is not s-monotone, then the object would be going backwards on occasion, for which the pusher would have to pull. 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 is then fairly simple to find an arbitrary s-monotone path in linear time, by following cell boundaries (which are s-monotone).

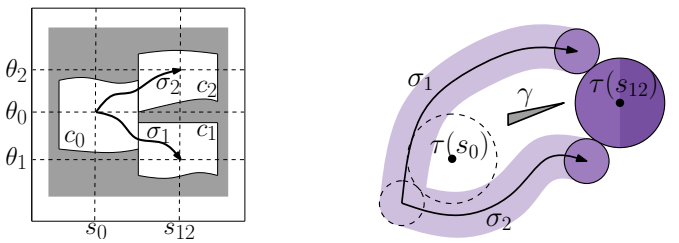


Fig. 5. Hypothetical situation in which a configuration-space cell would have an out-degree greater than one.

Such a path can lead to a lot of unnecessary motion for the pusher, and it is tempting to instead compute a Euclidean shortest path through the reachable cells. This would also yield an s-monotone path, but not necessarily one that minimizes the pusher's movement in the work space. We can circumvent this problem by performing our computations in the work space, instead of in the configuration space.

3.1. *Cells 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 call this region the corresponding *work-space cell*. All work-space cells glued together by their common boundaries form the region that P may move in to accomplish O's desired motion, provided we consider non-adjacent cells to be on a different "layer". We cannot just take the union of the cells, as P could then take a shortcut and lose O along the way. Figure 3 shows such a situation: moving through the union, P could push O to the lower-right corner, then leave it there and move on towards the top left. This way P always stays within the union of the work-space cells, but still fails to push O to its destination.

Lemma 1. *All cells in the cell graph have an out-degree of at most one.*

Proof. Suppose cell c_0 has out-edges to cells c_1 and c_2. From any $(s_0, \theta_0) \in c_0$ there must then be a push plan σ_1 to any $(s_{12}, \theta_1) \in c_1$ and a push plan σ_2 to any $(s_{12}, \theta_2) \in c_2$, as in Fig. 5(a). Property (W1) of well-behaved path sections then implies that any obstacle γ that causes c_1 and c_2 to be separate cells must be in the region between the areas swept out by the pusher along σ_1 and σ_2, and the area occupied by the object positioned at $\tau(s_{12})$, as in Fig. 5(b). But because the pusher maintains contact with the object at all times, this region must lie entirely within the area swept out by the object, which we assumed was obstacle free. □

3.2. *Computing a shortest contact-preserving push plan*

The union of the shortest paths from a point p inside a simple polygon to all the vertices of the polygon forms a tree. Given a triangulated simple polygon, this *shortest-path tree* can be computed by a linear-time algorithm due to Guibas *et al.*[15] The trivial shortest-path tree for the triangle containing p is extended triangle by triangle until it spans the whole polygon. García-López and Ramos[16] give a similar algorithm for a *pseudo-triangulated* simple *splinegon*, an analogue of a polygon using convex curves as edges. This can be used to yield a linear-time algorithm for shortest contact-preserving push plans through a given configuration space, as explained in the proof below.

To not have to make a case distinction between high and low obstacle density, we let q denote the maximal complexity of the configuration space for one path section. The unreduced configuration space has $q = O(n)$, while the reduced configuration space under low obstacle density has $q = O(1)$. Note, though, that a shortest push plan through the reduced configuration space is not necessarily as short as one through the unreduced configuration space. Hence $q = O(n)$ if we require a shortest push plan.

Theorem 4. *Given a k-section configuration space with complexity $O(q)$ per path section, a shortest contact-preserving push plan (of complexity $O(kq)$) through this space can be computed in $O(kq)$ time and space.*

Proof. A work-space cell w is the area $\mathcal{A}$ swept out by the push range over a piece of the object's path, minus the positions where the pusher would intersect any obstacles. Because of the way configuration-space cells are defined by a vertical decomposition, at most two obstacles can influence the shape of one work-space cell. Thus w is the set difference of $\mathcal{A}$ with at most two capsules. By property (W2) of well-behaved path sections, $\mathcal{A}$ is bounded by a constant number of convex, constant-complexity curves, so w must be as well. Thus a work-space cell is a splinegon with $O(1)$ vertices, which can be pseudo-triangulated in $O(1)$ time.[16] Figures 6(a) and 6(b) show an example of a work space and its work-space cells.

By Lemma 1, the reachable configuration space is a linear chain of $O(kq)$ cells. Having partitioned each cell into $O(1)$ pseudo-triangles, we then use the algorithm of García-López and Ramos[16] to compute the shortest-path tree of the work-space cells (pseudo-triangle by pseudo-triangle) in $O(kq)$ time and space. Figure 6(c) shows the resulting shortest-path tree for the example above. From this tree we can then extract the shortest path from the pusher's (black) starting position to its (bold) goal arc. This takes time proportional to the number of vertices on the path, which is $O(kq)$. □

4. Pushing and Releasing

Until now we've assumed the pusher can maintain contact with the object at all times. However, the situation depicted in Figs. 7(a) and 7(b) does not admit such

contact-preserving push plans. (In fact, this is true for *any* object path with the same start and end point, as proven in Appendix A.) It does admit an unrestricted push plan, as can be seen in Figs. 7(b) and 7(c).

4.1. *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 of a configuration-space obstacle (where points of

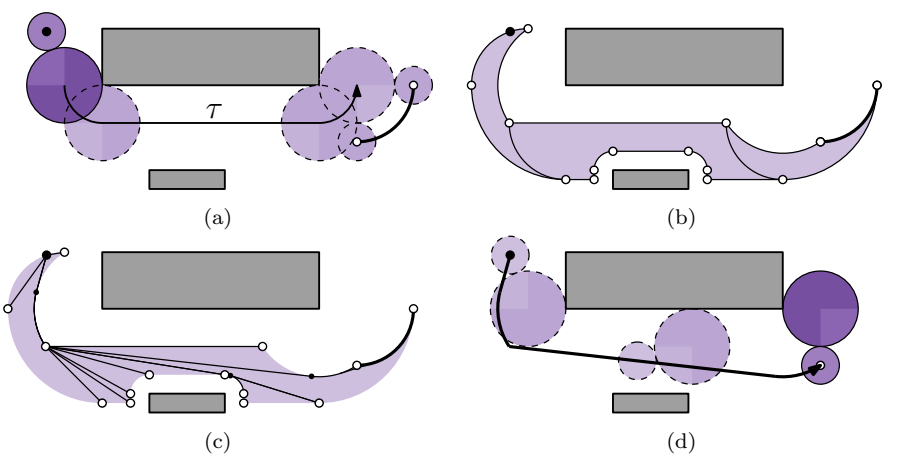


Fig. 6. (a) An example work space, and (b) its corresponding work-space cells. The pusher's starting point is drawn black, and its destination curve is drawn fat. (c) The final step in constructing (b)'s shortest-path tree. (d) The resulting (optimal) push plan.

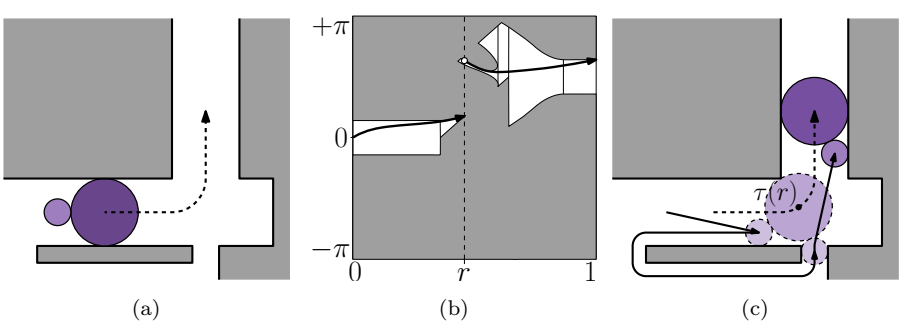


Fig. 7. (a) An example work space for which no contact-preserving push plan exists, (b) its configuration space, and (c) an unrestricted push plan for it, doing one release at position $\tau(r)$.

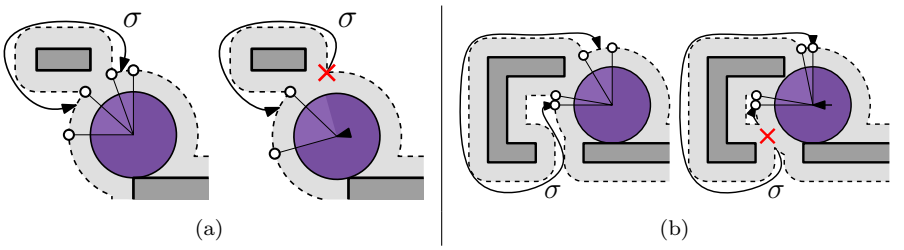


Fig. 8. The two situations in which moving the object backwards along τ would make us unable to maintain the pusher path σ.

vertical tangency are also considered vertices). We call the resulting set of $O(kq)$ positions (where $O(q)$, again, is the complexity of the configuration space of a path section) the *canonical releasing positions*. It suffices to check only these positions, as expressed by the following lemma:

Lemma 2. *If an unrestricted push plan exists for a given input, then there is also an unrestricted push plan where all releases happen at canonical releasing positions as defined above.*

Proof. Suppose we have an unrestricted push plan with one or more releases that don't happen at canonical releasing positions. Let $\tau(r)$ be an object position at which such a release happens, and σ be the path that the pusher follows from the position where it releases the object to the position where it recontacts. Because r is not a canonical releasing position there must be a canonical releasing position $r' < r$ with no other canonical releasing positions in between r' and r.

Suppose the releasing point for σ lies in cell c_1 of the free space and the recontact point in cell c_2. Now imagine moving the object backwards along τ from $\tau(r)$ to $\tau(r')$. In doing this we want to adjust our push plan so it remains valid, making it move a little less through cell c_1, a little more through cell c_2, and adjusting path σ accordingly for its new endpoints.

There are two ways in which this could fail: either the interval of valid pusher positions in which one of the endpoints of path σ resides vanishes, as in Fig. 8(a), or path σ gets cut off between the obstacles and the object, as in Fig. 8(b). The former can only happen at a vertex of c_1 or c_2, and the latter can only happen at a vertex of some configuration-space obstacle $\mathcal{C}_\gamma$. But such points would then form canonical releasing points between r and r', contradicting our assumption. □

4.2. *Computing an unrestricted push plan*

Restricting ourselves to canonical releasing positions, we cannot guarantee a shortest push plan anymore, so we abandon the work-space-cell approach and instead work with the cell graph directly. The cell graph encodes which configurations are reachable from which other configurations, assuming the pusher and object

maintain contact. By adding edges, we'll transform this into the *extended cell graph*, which encodes the same connectivity information when the pusher *can* let go of the object. To determine whether an edge between two cells should be added we have to solve a standard path-planning problem for the pusher, with the stationary object being an additional obstacle. The algorithm in more detail is as follows:

(1) Compute a *road map* $\mathcal{S}$ for P among the obstacles.[1]
(2) At each canonical releasing point r:
   (a) Add O positioned at $\tau(r)$ as an extra obstacle in $\mathcal{S}$ to get $\mathcal{S}_r$.
   (b) Determine the set C_r of cells intersected by a vertical line through r.
   (c) Determine for each cell in C_r to which component of $\mathcal{S}_r$ its pusher positions belong.
   (d) Add edges in the cell graph between cells sharing a component of $\mathcal{S}_r$.
(3) Compute a path through the extended cell graph.
(4) Convert the path into a push plan. For edges of the original cell graph this is straightforward, for every extra edge use the respective $\mathcal{S}_r$ to find a path for P.

To construct the initial road map (Step 1) we take the union of the capsules $\gamma \oplus D(r_p)$ for $\gamma \in \Gamma$, and then construct a vertical decomposition of its complement. The union can be done in $O(n \log^2 n)$ worst-case time (using an algorithm by Kedem *et al.*[12]), or in $O(n \log n)$ expected time (using an algorithm by Miller and Sharir[13]). After this preprocessing, the time taken by the remaining steps is expressed by the following theorem:

Theorem 5. *Given a road map for P among the obstacles, and a configuration space with complexity $O(q)$ per path section, an unrestricted push plan (of complexity $O(kqn)$) can be computed in $O(kq \log(kq) + kq^2 \log n + kqn)$ time using $O(kqn)$ space.*

Proof. While computing the configuration space, all $O(kq)$ vertices defining canonical releasing positions were already computed, so this takes no extra time. Steps 2, 3, and 4 can then be done in the stated bounds using standard techniques for point location (Steps 2(b) and 2(c)), and depth-first search (Step 3). □

5. Conclusion

We have studied the manipulation path-planning problem of a disk-shaped pusher moving a disk-shaped object along a given path, among non-intersecting line segments in the plane. We looked at the case where the pusher and object must maintain contact, as well as the case where there is no such restriction. For the contact-preserving case, we improved the running time of the only known algorithm, and gave the first algorithm to compute a shortest push plan. For the unrestricted case, we gave the first algorithm to compute a push plan at all. (The running times of these algorithms were summarized in Table 1 in the introduction.) Our algorithms also handle a more general class of input paths than prior work, and can

be modified to handle a more general class of obstacles as well (specifically, convex pseudodisks).

An obvious open question is how to compute a shortest unrestricted push plan, or if the running times of our algorithms can be further improved. Additionally, one may be interested in other shapes of the pusher and/or object, but a pushing motion may then rotate either or both of them. The respective orientations of the pusher and object will make the configuration space higher dimensional. Lynch and Mason[17] discuss conditions under which their relative orientation remains fixed, making the problem somewhat more tractable, but our method based on a 2-dimensional configuration space would still not suffice.

Applying the configuration-space approach to the problem where only a destination for the object is given (rather than a path), is also non-trivial. While there is a straightforward analogue to our configuration-space obstacles in this 3-dimensional configuration space, there is none for our forbidden push ranges. It may be possible to use some form of constrained path finding instead, but we have not explored this possibility.

References

1. J. C. Latombe, *Robot Motion Planning* (Kluwer Academic Publishers, 1991).
2. M. T. Mason, *Mechanics of Robotic Manipulation* (MIT Press, 2001).
3. K. Y. Goldberg, Orienting polygonal parts without sensors, *Algorithmica* **10**(2–4) (1993) 210–225.
4. H. Arai and O. Khatib, Experiments with dynamic skills, *Proc. Japan-USA Symp. Flexible Automation* (1994), pp. 81–84.
5. M. T. Mason and K. M. Lynch, Dynamic manipulation, *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems* (1993), pp. 152–159.
6. D. Nieuwenhuisen, A. van der Stappen and M. Overmars, Pushing a disk using compliance, *IEEE Trans. Robotics* **23**(3) (2007) 431–442.
7. S. M. LaValle and J. J. Kuffner, Rapidly-exploring random trees, *Algorithmic and Computational Robotics: New Directions*, eds. B. R. Donald, K. M. Lynch and D. Rus (2001), pp. 293–308.
8. M. de Berg, M. J. Katz, A. F. van der Stappen and J. Vleugels, Realistic input models for geometric algorithms, *Algorithmica* **34**(1) (2002) 81–97.
9. P. K. Agarwal, J.-C. Latombe, R. Motwani and P. Raghavan, Nonholonomic path planning for pushing a disk among obstacles, *Proc. IEEE Int. Conf. Robotics & Automation*, Vol. 4 (1997), pp. 3124–3129.
10. D. Nieuwenhuisen, Path planning in changeable environments, Ph.D. dissertation, Universiteit Utrecht, The Netherlands (2007).
11. M. A. Peshkin and A. C. Sanderson, Minimization of energy in quasi-static manipulation, *IEEE Trans. Robotics & Automation* **5**(1) (1989) 53–60.
12. K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discr. Comput. Geom.* **1** (1986) 59–70.
13. N. Miller and M. Sharir, Efficient randomized algorithm for constructing the union of fat triangles and of pseudodiscs, unpublished manuscript (1991).
14. I. J. Balaban, An optimal algorithm for finding segment intersections, *Proc. 11th ACM Symp. Comput. Geom.* (1995), pp. 211–219.

15. L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica* **2**(1) (1987) 209–233.
16. J. García-López and P. A. Ramos, A unified approach to conic visibility, *Algorithmica* **28**(3) (2000) 307–322.
17. K. M. Lynch and M. T. Mason, Stable pushing: Mechanics, controllability, and planning, *Int. J. Robotics Res.* **15**(6) (1996) 533–556.

Appendix A. Proof that Releasing can be Necessary

In Fig. 7(a) we saw a work space which permits no contact-preserving push plan for the given object path. One may wonder whether this object path is merely ill-chosen, and whether a different object path *would* admit a contact-preserving push plan. We will prove that this is not the case.

Assume, without loss of generality, that $r_o = 1$ and $r_p = \mu$, with $0 < \mu < 1$. Our goal is to get the object from the lower left part of Fig. 9(a), around the bend, to the upper right part of the figure. The path τ which the object then takes will have to go through the collision-free area depicted in Fig. 9(b).

Because the horizontal and vertical corridor are exactly as wide as the object, the object can only move through them in one way (and this is easily accomplished by the pusher). We'll therefore consider the intermediate section of the object's path connecting these two straight-line sections, that is, the subpath from point b to point e in Fig. 9(b).

In particular, we'll look at the *highest* possible such subpath, that is, the subpath that maximizes the object's distance traveled in the positive y direction for the distance traveled in the positive x direction. This motion is achieved by minimizing the pusher's y-coordinate at all times. Thus the pusher should start by sliding compliantly along the bottom obstacle. This will turn the object on a circular arc around v_L, but it can't always reach e via this arc, as stated in the following lemma:

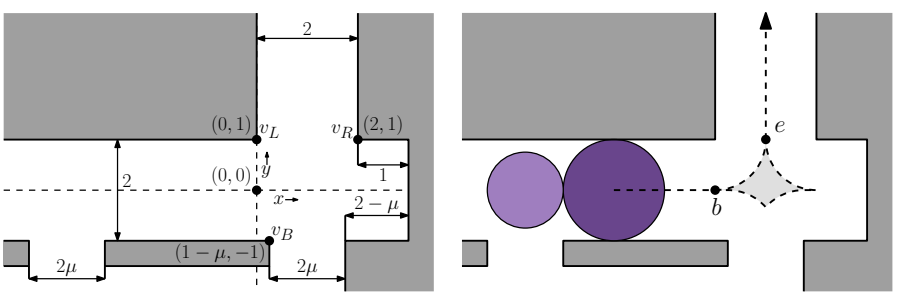
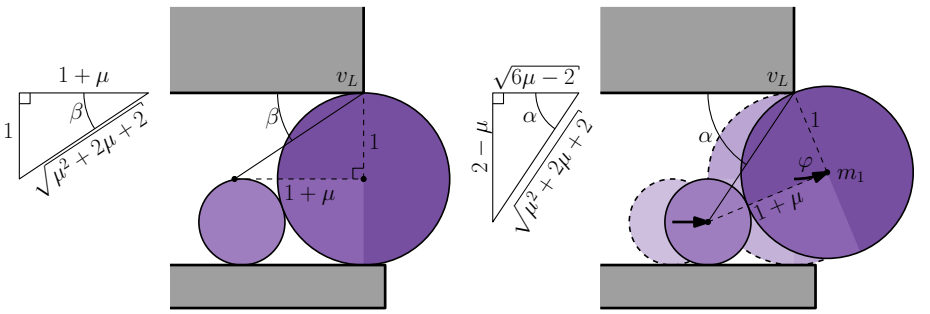


Fig. 9. (a) The environment with the coordinate system that we'll use. Measurements assume that $r_o = 1$ and $r_p = \mu$. (b) The initial situation, and the area through which the object's center can move without intersecting any obstacles.



(a) (b)

Fig. 10. (a) Initial position. (b) Position m_1 where O has turned by an angle $\varphi = a - b$ around v_L and P can no longer turn it further. This happens when $\mu > 1/3$.

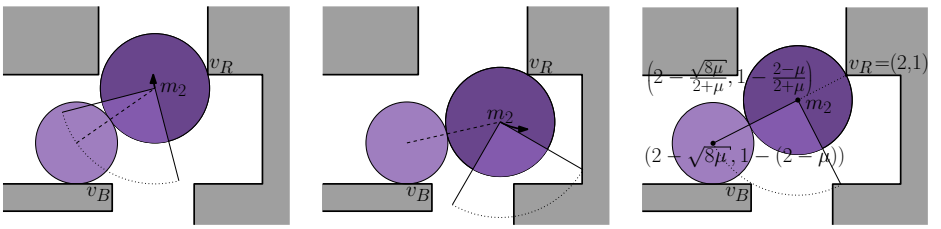
Lemma 1. *In the example of Fig.* 9*, the pusher cannot push the object compliantly around v_L all the way to e if $\mu > 1/3$.*

Proof. In this motion, the object's position describes a circular arc with radius r_o around v_L. To keep the object going along this arc, the ray from the pusher's center to the object's center must intersect the arc, either properly or tangentially. So if there is a position of the object along the arc such that the pusher cannot push tangentially to the arc, the motion cannot be completed. Figure 10 shows that this happens when $\sqrt{\mu^2 + 2\mu + 2} > 2 - \mu$. Squaring both sides of the equation and simplifying yields $\mu > 1/3$. □

From now on, assume that $\mu > 1/3$. Lemma 1 then implies that if the pusher tries to push the object compliantly around v_L, the object will end up at some point m_1 between b and e. From b to m_1, the object will have turned by some angle $\varphi < \pi/2$ around v_L, where (see Fig. 10):

$$\begin{aligned} \varphi &= \alpha - \beta \\ \sin(\alpha) &= \frac{2-\mu}{\sqrt{\mu^2+2\mu+2}}, \qquad \cos(\alpha) = \frac{\sqrt{6\mu-2}}{\sqrt{\mu^2+2\mu+2}} \\ \sin(\beta) &= \frac{1}{\sqrt{\mu^2+2\mu+2}}, \qquad \cos(\beta) = \frac{1+\mu}{\sqrt{\mu^2+2\mu+2}}. \end{aligned} \tag{A.1}$$

Recall that θ is the pushing angle, that is, the angle that the line from P's to O's center makes with the positive x-axis. If the pusher continues its straight line along the bottom obstacle, the object will move away from v_L to the right and upwards, thus increasing θ. The resulting motion was studied by Agarwal *et al.*,[9] and they refer to it as a *hockey-stick curve*. As a function of θ, this motion is given by the



(a) (b) (c)

Fig. 11. Being pushed along the hockey-stick curve, the object will eventually hit the outer corner vertex v_R. (a) If the object hits v_R high enough, the pusher has no trouble completing the path for the object. (b) But if the object hits v_R too low, the pusher can't do anything but push the object into the pocket hole. (c) The cut-off point is when the line from the pusher's center through m_2 passes through v_R.

following equations, where $\varphi \leq \theta \leq \pi/2$:

$$x(\theta) = (1+\mu)\ln\left(\frac{\tan(\theta/2)}{\tan(\varphi/2)}\right) + (1+\mu)\Big(\cos(\theta) - \cos(\varphi)\Big) + \sin(\varphi)$$
$$y(\theta) = (1+\mu)\sin(\theta) - 1 + \mu\,. \tag{A.2}$$

This motion also won't always get the object to e, as stated in the following lemma:

Lemma 2. *Following the hockey-stick curve from m_1 on, the object will eventually hit vertex v_R, at some point m_2. When m_2 is on or below the line $y = 1 - \frac{2-\mu}{2+\mu}$, the only possible continuations of the object's path lead it into the pocket hole on the right, and e cannot be reached.*

Proof. The curvature of the hockey-stick curve is smaller than that of the circular arc from b to e around v_L. Thus the object will reach the horizontal line through e at a point to the right of e itself. But by then point v_R would have already passed into the object's interior. Hence the object inevitably bumps into v_R at some point m_2 along the hockey-stick curve.

The y-coordinate of m_2 uniquely determines the angle θ at which the pusher can continue pushing from its contact with the bottom obstacle, as well as the push range which would push the object upwards compliantly on a circular arc around v_R. If m_2 is high enough, θ may lie in this push range, as seen in Fig. 11(a).

If m_2 is too low, θ will be outside of the needed push range. Having pushed the object against v_R, the pusher will still be to the left of vertex v_B, thus it can not get below the object. Therefore, the only remaining options for pushing lead the object into the pocket hole, as seen in Fig. 11(b).

The cut-off point where θ lies just outside the needed push range is when P's center, O's center, and vertex v_R are colinear. O's center then has coordinates $\left(2 - \frac{\sqrt{8\mu}}{2+\mu}, 1 - \frac{2-\mu}{2+\mu}\right)$, as illustrated by Fig. 11(c). □

We are now ready to formulate and prove the main theorem of this appendix.

Theorem 1. *There is a μ_c, $1/3 < \mu_c < 1$, such that for all $\mu > \mu_c$, the example of Fig. 9 admits no contact-preserving push plans for any of the possible object paths τ, while it does admit an unrestricted push plan (for some of these paths).*

Proof. From Eq. (A.2) it follows that the hockey-stick curve intersects the line $y = 1 - \frac{2-\mu}{2+\mu}$ exactly when:

$$\sin(\theta) = \frac{2-\mu}{2+\mu}\,. \tag{A.3}$$

The point on this line where the object touches v_R has x-coordinate $2 - \frac{\sqrt{8\mu}}{2+\mu}$. Thus v_R will be hit when the object is on or below the line $y = 1 - \frac{2-\mu}{2+\mu}$ if and only if:

$$x(\theta) \geq 2 - \frac{\sqrt{8\mu}}{2+\mu}\,. \tag{A.4}$$

Combining Eqs. (A.1) through (A.4) this condition becomes:

$$\ln(f(\mu)) \geq g(\mu)\,, \tag{A.5}$$

where:

$$f(\mu) = \frac{\left((2-\mu)\,(1+\mu) \;-\; \sqrt{6\mu-2}\right)\,\left(2+\mu-\sqrt{8\mu}\right)}{(2-\mu)\,\left(\mu\,(3+\mu) \;-\; (1+\mu)\,\sqrt{6\mu-2}\right)}$$

and

$$g(\mu) = \frac{\sqrt{6\mu-2}+2-\sqrt{8\mu}}{1+\mu}\,.$$

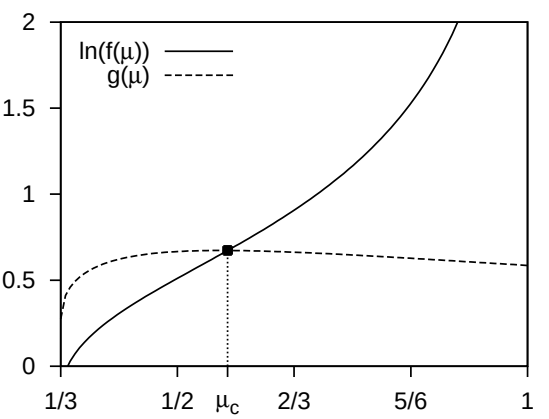


Fig. 12. The smallest μ value for which the example of Fig. 9 admits no contact-preserving push plan.

On the domain $\mu \in (1/3, 1)$, both g and the logarithm of f are continuous functions, and they intersect each other at exactly one $\mu = \mu_c$ (≈ 0.57173). For $\mu < \mu_c$ the function g has a greater value than the logarithm of f, and vice versa for $\mu > \mu_c$, as shown in Fig. 12. Thus there is a μ_c such that, for all $\mu > \mu_c$, the highest possible path for the object hits v_R at a point too low to escape the pocket hole. Since any other path is below this highest possible path, no contact-preserving push plans can exist.

In contrast, for some object paths an unrestricted push plan exists for all μ between 0 and 1, including those above μ_c. Figure 13 illustrates one such object path and the resulting unrestricted push plan. □

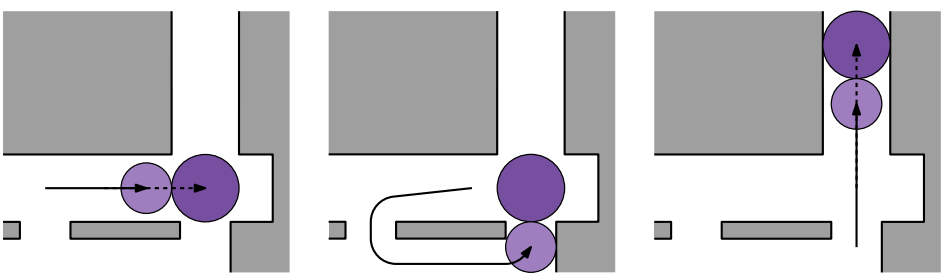


Fig. 13. When the pusher is allowed to release the object, a push plan *does* exist.