2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Computing Push Plans for Disk-Shaped Robots

Mark de Berg Dirk H.P. Gerrits
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, The Netherlands
E-mail: mdberg@win.tue.nl, dirk@dirkgerrits.com

***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. 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.**

I. Introduction

A fundamental problem in robotics is *path planning* [15], 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* [12] 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 [12], squeezing [7], rolling [2], and even throwing [13].

The manipulation path-planning problem studied here involves *pushing* [12]. 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. [15]–[17] developed a probabilistically complete algorithm for this based on the *Rapidly-exploring Random Trees* path-planning algorithm [10]. 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.

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*. 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*. (By "constant complexity" we mean that such a curve takes $O(1)$ space to represent, and for any two we can compute their $O(1)$ points of intersection in $O(1)$ time.) 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. [15], [18] also developed a subroutine that solves the problem just described. They assume the object path consists only of line segments and circular arcs, 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. If one assumes low obstacle density [5], 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. [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-size pusher but not for our disk-shaped pusher: there may be obstacles in the way.

Our results: We present a new approach to compute push plans for disk-shaped robots, which 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 Nieuwenhuisen et al. Table I 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 do not need $O(n^2 \log n)$ time (nor $O(n^2)$ space) for preprocessing.

II. The configuration space

A general-purpose technique for path planning is to translate the problem from the *work space* into the *configuration space* [4]. The work space is the environment in which the robot has to find a path. A *configuration* is one specific

	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 contact-preserving push plan	$kn \log n$	$kn \log n$ $^{(*)}$	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
A shortest contact-preserving push plan	—	$kn \log(kn)$	—	$kn \log(kn)$
Any unrestricted push plan	—	$kn \log(kn) + kn^2 \log n$	—	$(k+n)\log(k+n) + kn$

$^{(*)}$ These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

TABLE I

A COMPARISON OF THE ASYMPTOTIC RUNNING TIMES OF NIEUWENHUISEN'S APPROACH AND OURS FOR COMPUTING CONTACT-PRESERVING AND UNRESTRICTED PUSH PLANS.

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 things 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 Sections III and IV.

Preliminaries: 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 contiguous 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 Fig. 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 Fig. 1(b).)

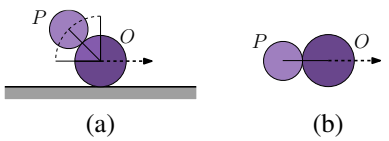

Fig. 1. The push range for (a) a compliant motion, and (b) a non-compliant motion.

The exact size of the push range for compliant motions depends on the friction characteristics of the two disks and the obstacles. Nieuwenhuisen [15] 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 assume (as do Nieuwenhuisen and Agarwal et al.) that pushing is *quasi-static* [19], i.e. when pushing stops, the object also stops instantly. Other than that, we abstract away from compliance and friction entirely. We assume that $\tau : [0,1] \to \mathbb{R}^2$ does not take the object through any of the obstacles, and that this path consists of k constant-complexity path sections $\tau_1, \ldots, \tau_k$ which are all *well-behaved*. A path section τ_i is well-behaved if all of the following hold (see Fig. 2 and 3):

(A1) We can compute the push range 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.)

(A2) The area swept out by the push range as the object moves along τ_i does not intersect itself, and is bounded by four convex, constant-complexity curves (the push ranges at either end, and the paths traced out by the two end points of the push range) which can be computed in $O(1)$ time.

(A3) There is a constant $d = O(r_o)$ such that, after the object has moved a distance d along the 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 (A2).

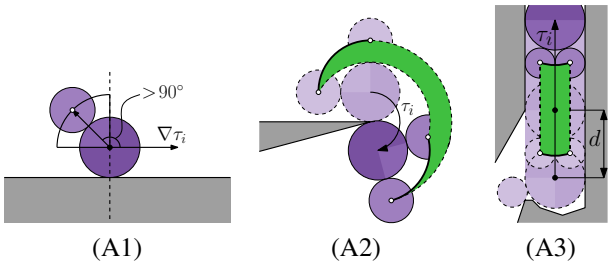

Fig. 2. The three properties satisfied by well-behaved path sections.

For correctness of our algorithms, (A1) and (A2) suffice, but (A3) allows us to derive better running times in case the obstacles are not too densely packed. The line segments and circular arcs used as path sections by Nieuwenhuisen et al. satisfy all three criteria, and are thus well-behaved.

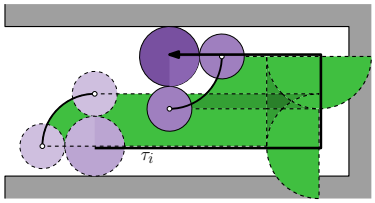

Fig. 3. A path section where well-behavedness property (A2) is *not* satisfied (the area swept out by the push range self-intersects, and its boundary curves are not convex). The path section *can* be broken up into three well-behaved path sections.

Shape of the configuration space: A configuration in our problem 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 maintain contact at all times—for now; we will lift this restriction in Section IV—, 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 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. An example is shown in Fig. 4(a)–(b).

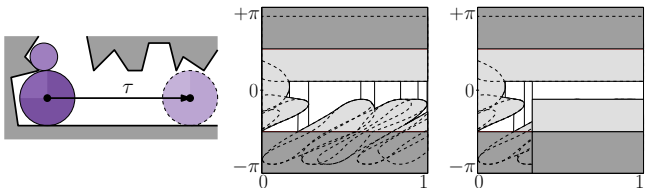

Fig. 4. (a) An example work space, (b) its configuration space, and (c) its reduced configuration space (defined below). The s-axis is horizontal, the θ-axis is vertical. Configuration-space obstacles are drawn dashed in light gray, the forbidden push range is drawn in dark gray.

Theorem 1: The configuration space for each path section has complexity $O(n)$ (i.e. 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 $\gamma \in \Gamma$. We will prove that $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. It then follows that $\text{FPR}_i \cup \bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$, 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 an intersection of the corresponding capsules. These capsules form a collection of *pseudodisks* [4], 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 constant-complexity curve. 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 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. Step 3 can be performed by a deterministic algorithm by Kedem et al. [9] that uses $O(n \log^2 n)$ time, or a randomized incremental algorithm by Miller and Sharir [14] that uses $O(n \log n)$ expected time, both using $O(n)$ space. For Step 4 and 5 we extend vertical lines from each of the $O(n)$ vertices of the forbidden space to get a vertical decomposition of the free space into cells. These can be computed and connected together with a sweep-line algorithm in $O(n \log n)$ time and $O(n)$ space. ■

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 $\text{length}(\gamma) \geq \text{diam}(D)$. By property (A3) of well-behaved 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 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 (i.e. $O(k)$ in total), and can be computed in $O((k+n)\log(k+n))$ time using $O(k+n)$ space.

Proof: From the above discussion it follows that the reduced configuration space for one path section has $O(1)$ complexity. Computing all $\mathcal{C}_{\gamma,i}$ and FPR_i section-by-section and obstacle-by-obstacle would still take $\Omega(kn)$ time, though. Instead, we can compute them for all path sections and obstacles at once, using an algorithm by Balaban [3] for computing intersections. This takes $O((k+n)\log(k+n))$ time and $O(k+n)$ space. The remaining work takes $O(1)$ time per section. ■

III. 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's then fairly simple to find an arbitrary s-monotone path in linear time, by following cell boundaries (which are s-monotone).

It is tempting to instead compute a Euclidean shortest path through the reachable cells. This would 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.

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 could 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. (That is, if 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.)

Lemma 1: All cells in the cell graph have an outdegree of at most one.

Proof: Suppose cell c_0 has outedges 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 (A1) of well-behaved path sections then implies that any obstacle 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. ■

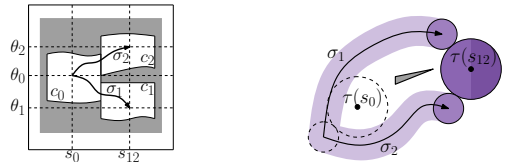

Fig. 5. Hypothetical situation in which a configuration-space cell would have an outdegree greater than one.

Computing a shortest contact-preserving push plan: Guibas et al. [8] presented a linear-time algorithm to compute the shortest-path tree of a triangulated simple polygon. Because of Lemma 1, the ideas behind this algorithm can also be used to find a shortest path through the work-space cells of 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 a path section. For high obstacle density $q = O(n)$, and for the reduced configuration space under low obstacle density $q = O(1)$. (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 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 $O(kq\log(kq))$ time using $O(kq)$ space.

Proof: A work-space cell is the area swept out by the push range over a piece of the object's path, minus the areas where the pusher would intersect an obstacle. Because of the way configuration-space cells are constructed by a vertical decomposition, at most two obstacles can be involved in this. Thus, by property (A2) of well-behaved path sections, a work-space cell is a region bounded by a constant number of convex, constant-complexity curves. Fig. 6(a)–(b) shows an example of a work space and its work-space cells.

Now suppose $c_1, \ldots, c_m$ is the chain of work-space cells from the starting configuration to a destination configuration, and that we have computed the shortest-path tree for the vertices and curves of work-space cells $c_1, \ldots, c_i$. Let $\sigma_{i,1}$ and $\sigma_{i,2}$ be the shortest paths to the (at most) two vertices of c_i on the curve where it touches c_{i+1}. For the shortest-path tree for $c_1, \ldots, c_{i+1}$ we just need to find the shortest paths to the vertices and curves of c_{i+1}. Each of these will follow one of $\sigma_{i,1}$ or $\sigma_{i,2}$, and then end in a tangent line to this path, or a bitangent of this path and one of c_{i+1}'s bounding curves (possibly followed by the rest of this curve). Fig. 6(c) shows an example. The point until which $\sigma_{i,1}$ or $\sigma_{i,2}$ is followed can be found by binary search. Since there are $O(kq)$ cells, we need to do $O(kq)$ such searches, using $O(\log(kq))$ time each, to compute the shortest-path tree of the work-space cells. ■

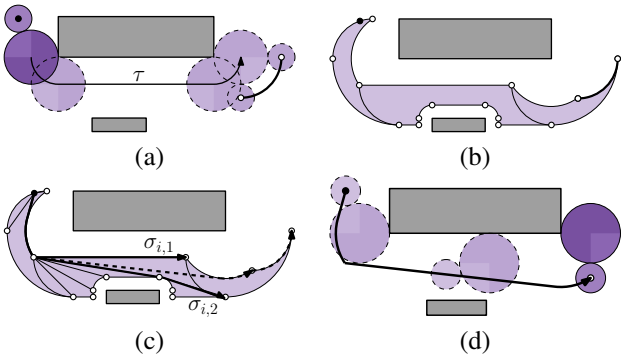

Fig. 6. (a) An example work space, and (b) its corresponding work-space cells. The pusher's starting point is drawn black, and its destination curve is drawn fat. (c) The final step in constructing (b)'s shortest-path tree. (d) The resulting (optimal) push plan.

IV. Pushing and Releasing

Until now we've assumed the pusher can maintain contact with the object at all times. However, the situation depicted in Fig. 7(a)–(b) does not admit such contact-preserving push plans. (In fact, we've proven [6] 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 Fig. 7(b)–(c).

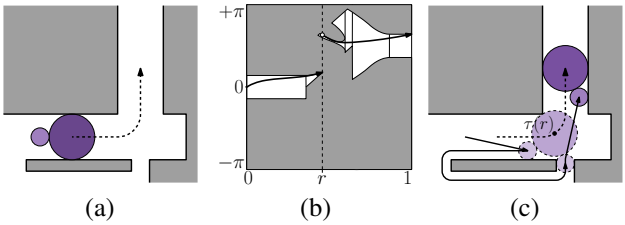

Fig. 7. (a) An example work space for which no contact-preserving push plan exists, (b) its configuration space, and (c) an unrestricted push plan for it, doing one release at position $\tau(r)$.

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 $O(kq)$ 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 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 release 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 be canonical releasing points between r and r', contradicting our assumption. ■

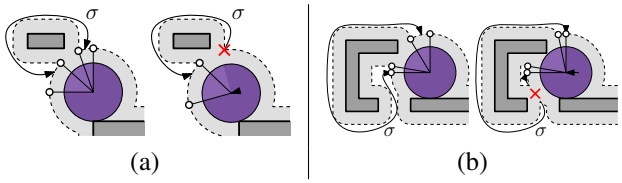

Fig. 8. The two situations in which moving the object backwards along τ would make us unable to maintain the pusher path σ.

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 this connectivity information assuming the pusher *can* let go of the object when necessary. 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 [4].
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 the 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 and then make a vertical decomposition of their complement. The union can be done in $O(n \log^2 n)$ worst-case time (using an algorithm Kedem et al. [9]), or in $O(n \log n)$ expected time (using an algorithm by Miller and Sharir [14]). 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 to 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). ■

V. 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 I 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 [11] 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 not so 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] 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*, vol. 4, 1997, pp. 3124–3129.

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

[3] I. Balaban, "An optimal algorithm for finding segment intersections," in *Proc. 11th ACM Symp. Comput. Geom.*, 1995, pp. 211–219.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008, ch. Chapter 13: Robot Motion Planning.

[5] M. de Berg, M. Katz, A. van der Stappen, and J. Vleugels, "Realistic input models for geometric algorithms," in *Proc. 13th ACM Symp. Comput. Geom.*, 1997, pp. 294–303.

[6] D. Gerrits, "Designing push plans for disk-shaped robots," Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2008. [Online]. Available: http://alexandria.tue.nl/extra1/afstversl/wsk-i/gerrits2008.pdf

[7] K. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, vol. 10, no. 2–4, pp. 210–225, 1993.

[8] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside simple polygons," in *Proc. 2nd ACM Symp. Comput. Geom.*, 1986, pp. 1–13.

[9] K. Kedem, R. Livne, J. Pach, and M. Sharir, "On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles," *Discrete & Computational Geometry*, vol. 1, pp. 59–70, 1986.

[10] S. LaValle and J. Kuffner, "Rapidly-exploring random trees," in *Algorithmic and Computational Robotics: New Directions*, B. Donald, K. Lynch, and D. Rus, Eds., 2001, pp. 293–308.

[11] K. Lynch and M. Mason, "Stable pushing: Mechanics, controllability, and planning," *Int. Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, 1996.

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

[13] M. Mason and K. Lynch, "Dynamic manipulation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, 1993, pp. 152–159.

[14] N. Miller and M. Sharir, "Efficient randomized algorithm for constructing the union of fat triangles and of pseudodiscs," 1991, unpublished manuscript.

[15] D. Nieuwenhuisen, "Path planning in changeable environments," Ph.D. dissertation, Universiteit Utrecht, The Netherlands, 2007.

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

[17] ——, "Pushing a disk using compliance," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 431–442, 2007.

[18] ——, "Path planning for pushing a disk using compliance," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, 2005, pp. 4061–4067.

[19] M. Peshkin and A. Sanderson, "Minimization of energy in quasi-static manipulation," *IEEE Transactions on Robotics & Automation*, vol. 5, no. 1, pp. 53–60, 1989.