Designing Push Plans for Disk-Shaped Robots

Dirk H.P. Gerrits

28 May 2008

Master's Thesis

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

Supervisor: Prof. dr. Mark de Berg
Referees: Dr. Alexander Wolff
Prof. dr. ir. Jan Friso Groote

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 2
- 1.2 Global approach . . . 2
- 1.3 The studied subproblem . . . 3
- 1.4 Related work . . . 4
- 1.5 New contributions . . . 5

2 Preliminaries **6**
- 2.1 Friction and compliance . . . 6
- 2.2 Object motions for a straight-line push . . . 7
- 2.3 Pseudodisks and Minkowski sums . . . 8
- 2.4 Nieuwenhuisen's subroutine . . . 10

3 Pushing while maintaining contact **19**
- 3.1 Well-behaved path sections . . . 20
- 3.2 Shape of the configuration space . . . 22
- 3.3 Computing the configuration space . . . 26
- 3.4 Finding a contact-preserving push plan . . . 28
- 3.5 Finding a shortest contact-preserving push plan . . . 30
- 3.6 Low obstacle density . . . 32

4 Pushing and releasing **34**
- 4.1 Releasing can be beneficial . . . 34
- 4.2 Releasing can be necessary . . . 36
- 4.3 Canonical releasing positions . . . 40
- 4.4 Finding an unrestricted push plan . . . 42
- 4.5 Low obstacle density . . . 44

5 Conclusion **45**
- 5.1 Improvements over prior work . . . 45
- 5.2 Further research . . . 46

Bibliography **48**

Chapter 1

Introduction

A fundamental problem in robotics is *path planning* [15, Chapter 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 one of its most simple forms, the problem involves a point-shaped robot which can translate freely (not needing to accelerate or turn) in a 2-dimensional environment consisting of static, polygonal obstacles. From this simple basis, many more elaborate variants can be constructed. For example, one might want to consider:

- 3-dimensional (or higher dimensional) environments.
- Dynamic environments, in which the obstacles change shape, move, and/or turn.
- Environments which are not (fully) known to the robot beforehand, and still have to be explored.

The robot and its movement could also be more complex:

- The robot is usually assumed to have volume (as opposed to a point robot which has zero volume), for example being polygonal in shape. To avoid obstacles its motion might then have to include rotation as well as translation.
- The robot's movement might be similar to that of a car: only able to go forward or backward and having a non-zero turning radius (i.e. it cannot move sideways and it cannot turn on the spot).
- The robot may be just one of many robots in the environment, having to avoid not only the obstacles but the other robots as well.

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 [6], rolling [2], and even throwing [13].

This thesis studies one particular manipulation path planning problem involving *pushing* [12]. The robot, which will be referred to as the *pusher*, has to push a passive *object* to a certain destination. Both the pusher and the object are assumed to be circular disks, and they move amongst polygonal obstacles in the 2-dimensional Euclidean plane.

Nieuwenhuisen [15, 17, 18] studied this same problem, and developed an algorithm for it. After stating the problem in more detail we'll discuss his algorithm, which uses a subroutine to solve a related but simpler pushing problem. This thesis discusses the shortcomings of his subroutine, and develops a better one.

1.1 Problem statement

In the 2-dimensional Euclidean plane we are given the following:

- Two circular disks: the *pusher* P, and the *object* O. Each is defined by a radius (r_p and r_o, respectively), and an initial position ($\mathbf{s}_o \in \mathbb{R}^2$ and $\mathbf{s}_p \in \mathbb{R}^2$, respectively).
- A set $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ of non-intersecting line segments called the *obstacles*. That is, each edge of the "polygonal obstacles" mentioned earlier is seen as a separate obstacle.
- A *destination* position for the object ($\mathbf{g}_o \in \mathbb{R}^2$).

With the position of a disk we mean the position of its center. In figures we'll always draw the object in a darker shade than the pusher. Obstacles are drawn as thick black lines, and when the obstacles are in fact edges of polygons such polygons are colored light gray. (See Figure 1.1, for example.)

We are now interested in finding a *push plan*: a path for the pusher that will make it push the object from its initial position to its destination, with neither disk intersecting any obstacles along the way. Both disks *are* allowed to touch or slide along obstacles. In other words, when we say that a disk intersects an obstacle, we mean that its interior intersects the obstacle. Without loss of generality, we assume that the given initial and destination positions are such that the disks don't intersect any obstacles (if they did, no push plans would be possible).

If no such push plan exists we want to report that fact, otherwise we want to construct such a push plan. When multiple push plans exist we may be interested in finding the shortest one, i.e. the one minimizing the distance traveled by the pusher.

We follow Nieuwenhuisen in assuming that the pusher is smaller than the object (i.e. $r_p < r_o$). Without this assumption, the solution space is often vastly reduced, as the pusher will be more restricted in its movement than the object itself.

1.2 Global approach

In chapters 7 through 9 of his PhD thesis [15], Nieuwenhuisen gives a probabilistic algorithm for constructing these push plans. His method is based on the *Rapidly-exploring Random Trees* path-planning algorithm [10].

In a nutshell, this algorithm incrementally builds a tree of reachable placements for the object and pusher. Once the destination position of the object is connected to the tree (as a leaf), a push plan can be constructed by concatenating the push plans between the placements in the tree on a path from the root to this leaf. (See Figure 1.1.)

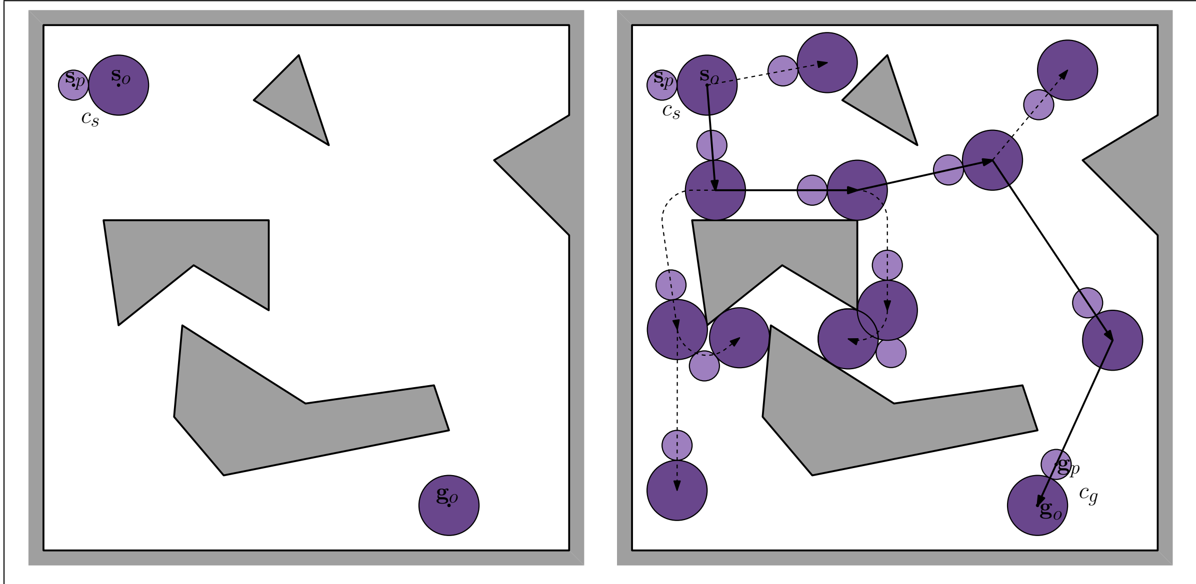


Figure 1.1: To create a push plan, Nieuwenhuisen's algorithm builds up a tree of placements for the object and pusher, with smaller push plans between them. The final push plan is then a concatenation of the smaller push plans from the tree's root to the leaf containing the object's destination.

The tree initially consists only of a root node containing the initial placement $c_s = (\mathbf{s}_o, \mathbf{s}_p)$ of the object and pusher. The tree is then iteratively extended by generating a random new object placement $\mathbf{r}_o$, finding the placement $c_n = (\mathbf{n}_o, \mathbf{n}_p)$ with the least distance between $\mathbf{n}_o$ and $\mathbf{r}_o$, and trying to construct a straight-line push plan between the two.

If this succeeds, with the pusher at some position $\mathbf{r}_p$, then $c_r = (\mathbf{r}_o, \mathbf{r}_p)$ is added to the tree as a child of c_n. If it fails, that means an obstacle was hit along the way, at some placement c_c. Then c_c (instead of c_r) is added as a child of c_n, and the obstacle boundaries are followed in both directions to create a subtree of placements reachable from c_c (see Figure 1.2).

Picking such random placements $\mathbf{r}_o$, and extending the tree with them, is continued until either a placement $c_g = (\mathbf{g}_o, \mathbf{g}_p)$ is added to the tree, or until some other stopping criterion is reached (say, the number of iterations exceeds some maximum). By occasionally picking $\mathbf{r}_o = \mathbf{g}_o$ it can be guaranteed that if a push plan exists, it will be found with high probability (given that enough iterations are performed).

1.3 The studied subproblem

As a subroutine, the algorithm just described needs a way to find push plans when a path for the object (through free space and/or along obstacle boundaries) is given, rather than just an

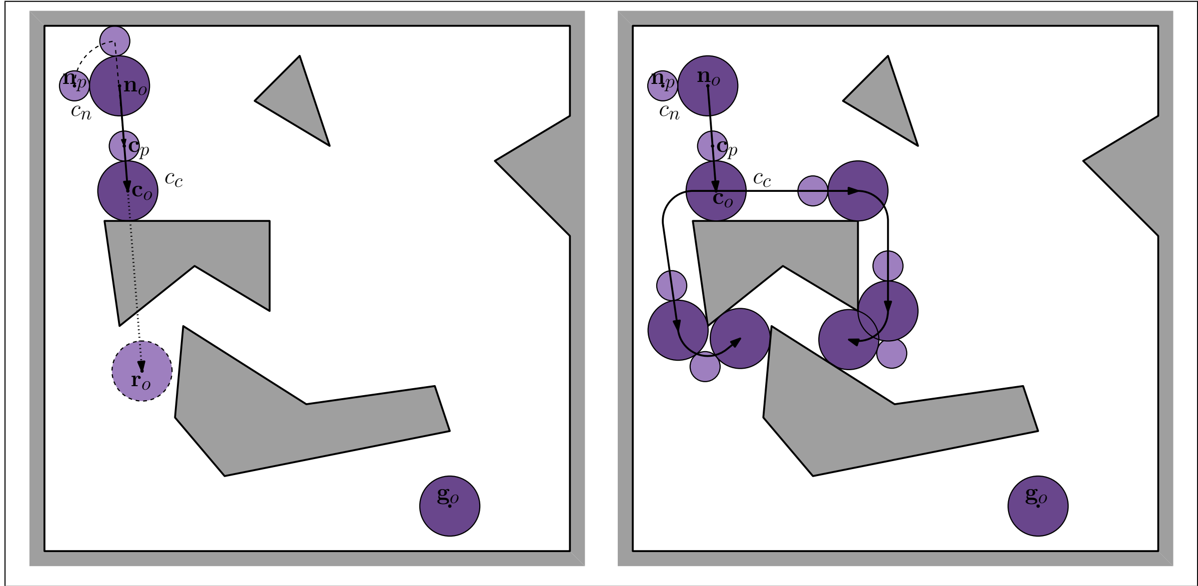


Figure 1.2: The algorithm tries to connect random object placements to the tree by attempting to find a push plan for moving the object to this new placement in a straight line. If this fails, placements along the obstacle boundaries are added to the tree instead.

initial and destination position. In other words we need a subroutine to solve the following subproblem:

Given:

- A circular *pusher* P (of radius r_p) and *object* O (of radius r_o with $r_p < r_o$), and their initial positions.
- A set $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ of non-intersecting line segments called the *obstacles*.
- A path τ for O from its initial position to some destination, in which O doesn't intersect any obstacles.

Find:

- A path for P (called a *push plan*) such that if P follows this path it will push O along τ as far as possible: either until the end of τ, or until pushing cannot continue (due to obstacles).

Solving this subproblem is the main topic of this thesis.

1.4 Related work

In addition to the global approach described above, Nieuwenhuisen also developed a subroutine for the subproblem where the object path is already given [15, 16]. The object path is assumed to be made up of only line segments and circular arcs, each of which is referred to as a *(path) section*. For n obstacles, and an object path consisting of k sections, his subroutine

constructs a push plan in $O(kn \log n)$ time (or $O((k+n)\log(k+n))$ under an assumption of low obstacle density [4]). However, it needs a hefty $O(n^2 \log n)$-time preprocessing step, uses $O(n^2+k)$ storage, and makes no attempt to find an optimal push plan. Furthermore, it is assumed that contact between the pusher and object can be maintained at all times, and the problem of constructing push plans that don't always maintain this contact is left to future research. We prove in Chapter 4 that there are indeed cases where contact cannot be maintained, and modify our own subroutine to handle this.

Agarwal et al. [1] studied the resulting motion of a unit disk when a point on its boundary is pushed along a straight line. By scaling this curve appropriately, it becomes exactly the motion of our object (which is not necessarily of unit radius) when our pusher (which is a disk rather than a point) moves in a straight line while maintaining contact with the object. This will be discussed in more detail in Section 2.2.

In the same paper, Agarwal et al. also described an algorithm to push their unit disk among polygonal obstacles from a given initial position to a given destination position. The algorithm discretizes the problem in two ways. Firstly, the angles that the line through the pushing point and the disk's center can make with the x-axis are discretized into $1/\varepsilon$ different values. Secondly, the combined boundary of the obstacles is sampled at m locations, and only those locations are considered as potential intermediate positions for the disk. The path-finding algorithm then runs in $O((1/\varepsilon)m(m+n)\log n)$ time, where n is the combined total number of vertices of the obstacles. There is a trade-off in that $1/\varepsilon$ and m have to be given high values to avoid missing a possible path, but for efficiency they should be given low values. 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.

1.5 New contributions

Chapter 2 first describes some concepts needed in later chapters, and discusses the results of Nieuwenhuisen and Agarwal et al. in some more detail.

Chapter 3 then presents a new method for computing *contact-preserving push plans*. In such push plans the pusher never releases contact with the object. Nieuwenhuisen's subroutine also computes such push plans, but our method is more general and allows for finding *shortest* push plans.

Chapter 4 discusses the limitations of only considering contact-preserving push plans, rather than *unrestricted push plans* where the pusher may release the object momentarily whenever it sees fit. Specifically, it gives examples where simple unrestricted push plans exist, but where contact-preserving push plans are either very complex, or don't exist at all. The approach of Chapter 3 is then extended to compute unrestricted push plans.

Chapter 5 summarizes our results and gives some concluding remarks and directions for the future.

Chapter 2

Preliminaries

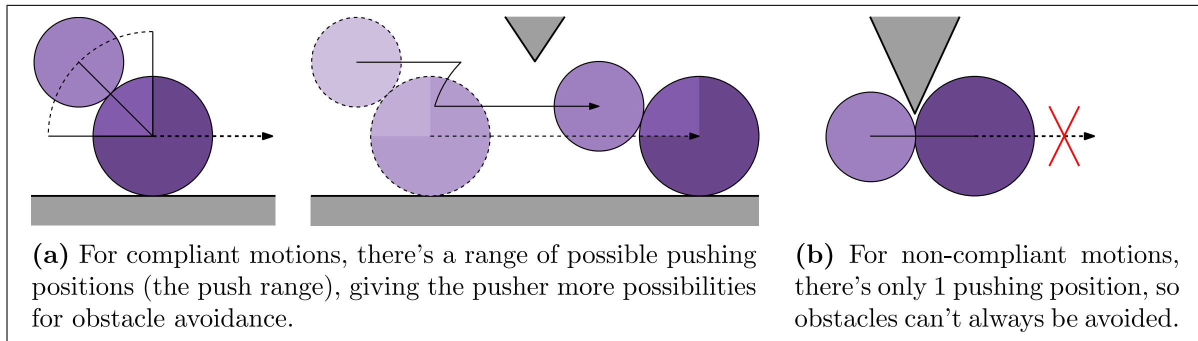
In this chapter we explore some aspects of the pushing problem a bit deeper, introducing needed concepts for later chapters. We also study the prior work on this topic by Agarwal et al. [1] and Nieuwenhuisen [15] in more detail. These works are guided by the physics of the real-world robotics problem of two disks pushing each other on a flat surface. When we present our new approach in Chapter 3 we'll abstract away from such considerations, leading to a more general method.

2.1 Friction and compliance

A *compliant motion* is one where the object slides along an obstacle. Such motions have several advantages over non-compliant motions. For one, they're 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 whole range of 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 2.1(a).)

With a non-compliant motion, the position to push from is determined entirely by the desired direction of motion for the object (i.e. the push range consists of a single pushing position). When the object's center moves on a straight line, for example, the pusher's center needs to stay on this same line. If any obstacles are in the way, there simply exists no push plan for that object motion. (See Figure 2.1(b).)

For compliant motions, the exact size of the push range depends on the friction characteristics of the two disks and the obstacles. (Maximally it is 90° wide (or 180° when the object is simultaneously compliant with two parallel obstacles) and this is what we'll use in our examples.) Friction also affects how pushing works for non-compliant paths. Nieuwenhuisen [15] assumes friction coefficients between the disks and between the object and obstacles are known, and shows how to compute the size of the push range from them. This push range is fixed, moving along with the object but never changing shape, as was illustrated in Figure 2.1(a). Furthermore, it is assumed that the disks don't slip during pushing, except during the motion described in Section 2.2. Lastly, pushing is assumed to be *quasi-static* [19]. This means that when pushing stops, the object also stops instantly. (This will never be the case in reality,



(a) For compliant motions, there's a range of possible pushing positions (the push range), giving the pusher more possibilities for obstacle avoidance.

(b) For non-compliant motions, there's only 1 pushing position, so obstacles can't always be avoided.

Figure 2.1: Compliant motions allow for a wider range of pushing positions than non-compliant motions, which aids in obstacle avoidance.

but it can be closely approximated by pushing slowly, or having very high friction between the disks and the floor.)

2.2 Object motions for a straight-line push

The problem we want to solve is to find the motion of the pusher that will accomplish a given motion of the object. To do that, we need to understand the reverse: what motion for the object results for a certain motion of the pusher? This was studied by Agarwal et al. for a pusher moving in a straight line [1]. They assumed a point-sized pusher, but our situation can be made equivalent to this by shrinking the pusher to radius 0 while growing the object to radius $r_o + r_p$. They also assumed that their object was of unit radius, but this simply means that we need to scale their formulas by a factor $r_o + r_p$.

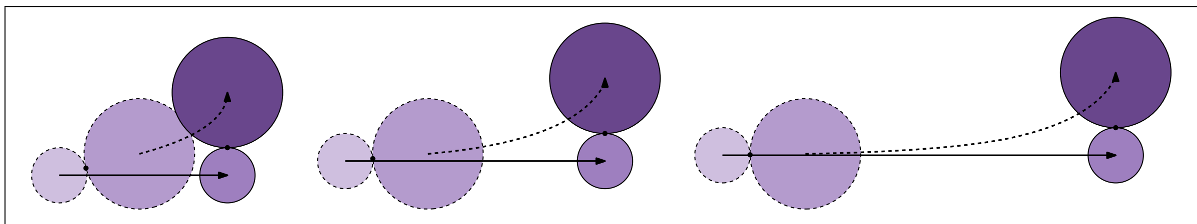


Figure 2.2: Some examples of the hockey stick curve motions of the object in response to a straight-line push for different pushing directions. (From left to right: $\varphi = \pi/12 = 15°$, $\varphi = \pi/36 = 5°$, and $\varphi = \pi/180 = 1°$.)

They found that the object motion curve resembles a *hockey stick*. Figure 2.2 shows some examples. In Cartesian coordinates, the curve is most easily parametrized by the angle θ that the line from the pusher's center through the object's center makes with the pushing direction. Starting from $\theta = \varphi$, this angle increases until the pushing direction becomes tangent to the object disk at $\theta = \pi/2$, and trying to push further would only make the pusher move away from the object.

Let's put the object's center at the origin, the positive x-axis in the pushing direction, and the y-axis such that the pusher's center has a negative y-coordinate. The object's motion for

$0 < \varphi \leq \theta < \pi/2$ is then described by [1]:

$$\begin{aligned} x(\theta) &= (r_o + r_p)\left(\ln\left(\frac{\tan(\theta/2)}{\tan(\varphi/2)}\right) + \cos(\theta) - \cos(\varphi)\right) \\ y(\theta) &= (r_o + r_p)\left(\sin(\theta) - \sin(\varphi)\right) \end{aligned} \tag{2.1}$$

For the limiting case of $\varphi = 0$, the motion of the object is a straight line (as would be expected).

We can also parametrize by time t. Assuming the pusher moves with unit velocity, the angle θ is related to time t as $\theta = 2\tan^{-1}\left(\tan(\varphi/2)\ e^t\right)$, where $0 \leq t \leq \ln(\cot(\varphi/2))$. The object's motion is then described by:

$$\begin{aligned} x(t) &= t + \frac{1 - \tan^2(\varphi/2)e^{2t}}{1 + \tan^2(\varphi/2)e^{2t}} - \cos(\varphi) \\ y(t) &= \frac{2\tan(\varphi/2)e^t}{1 + \tan^2(\varphi/2)e^{2t}} - \sin(\varphi) \end{aligned} \tag{2.2}$$

Another way to arrive at this same object motion is to consider the problem of pushing the object away from an obstacle it is compliant with. We want to achieve a distance of $2r_p$ between the object and the obstacle so that this obstacle no longer restricts the pusher, and in accomplishing this we want to minimize the distance traveled by the object. To do this, one should maximize the pushing angle θ, which happens when the pusher is compliant with the obstacle. By then pushing towards the object center by a small distance ε, the object and pusher move a bit further away from the obstacle, meaning that an even bigger pushing angle θ can be assumed. (See Figure 2.3.) Repeating this process gives a polygonal motion, that becomes the smooth hockey stick curve when ε goes to 0. The hockey stick curve (with maximal φ) is therefore the way of moving the object away from an obstacle with the least distance traveled parallel to the obstacle.

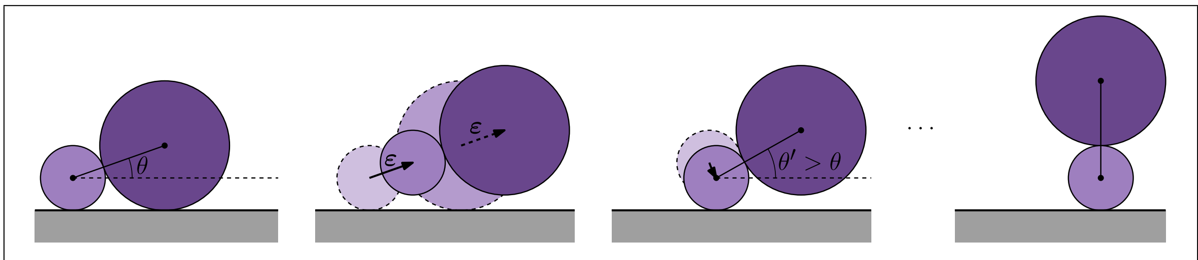


Figure 2.3: Moving the object away from an obstacle, in such a way that the distance traveled is minimized, is done by maximizing the pushing angle θ.

2.3 Pseudodisks and Minkowski sums

A pair of planar objects o_1 and o_2 is called a pair of *pseudodisks* when the sets $\partial o_1 \cap int(o_2)$ and $int(o_1) \cap \partial o_2$ are each connected. (Here ∂o is the boundary of o, and $int(o)$ is the interior of o.) Figure 2.4 shows some examples.

Pseudodisks possess the following useful properties [5, Chapter 13]:

- A pair of pseudodisks has at most two *boundary crossings.*

  A boundary crossing is an intersection point where the boundary of one pseudodisk crosses from the interior to the exterior of the other. (Figure 2.4 shows the boundary crossings of the pictured shapes as fat dots.)

- The complexity of the union of any set of pseudodisks is linear in the total complexity of the pseudodisks.

  For example: if the pseudodisks are polygonal, and together have n edges in total, then their union will have $O(n)$ edges.

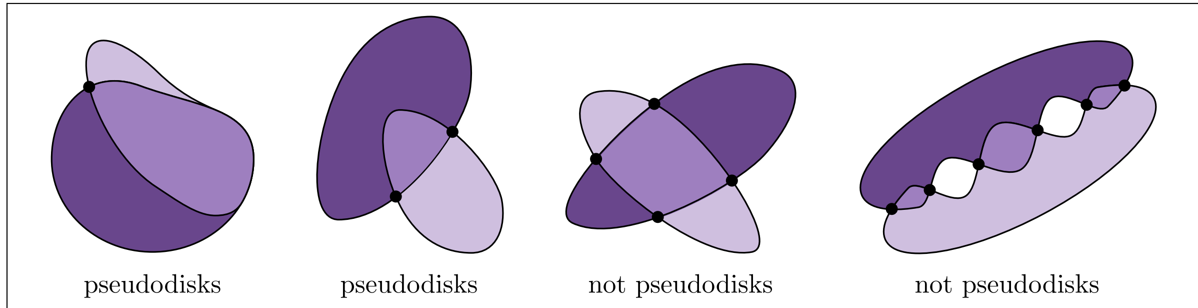
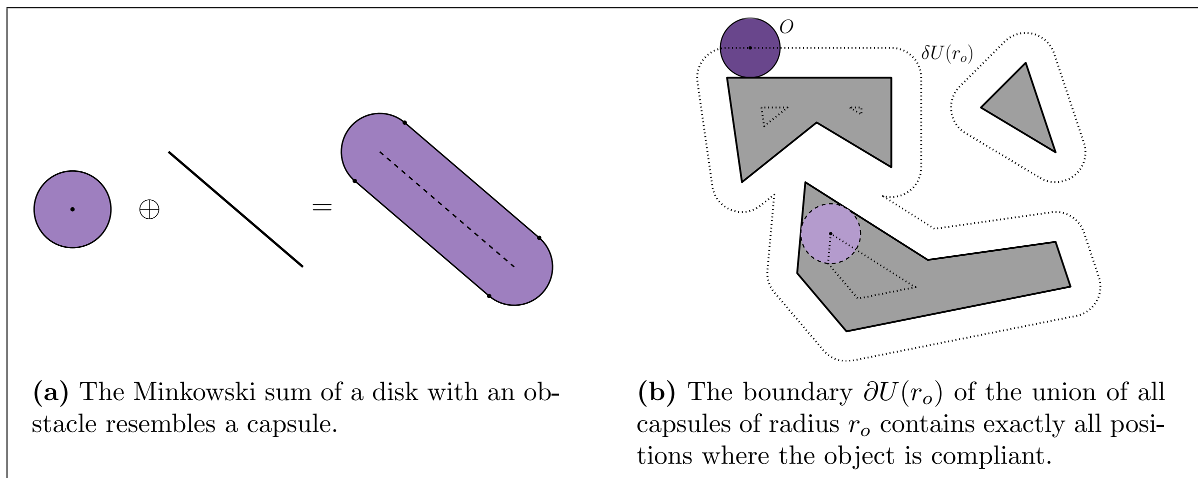


Figure 2.4: Some examples showing the concept of pseudodisks.

The particular pseudodisks helpful for our path-finding problem are the *Minkowski sums* of each of the obstacles with a disk. The Minkowski sum of two planar objects o_1 and o_2 is defined as $o_1 \oplus o_2 = \{\mathbf{x_1} + \mathbf{x_2} \mid \mathbf{x_1} \in o_1, \mathbf{x_2} \in o_2\}$. For an obstacle γ and a disk $D(r)$ of radius r the Minkowski sum $\gamma \oplus D(r)$ is a rectangle capped by two half-disks, resembling a *capsule* as illustrated in Figure 2.5(a). The interior of such a capsule consists of all the positions where the disk would intersect the obstacle if it were centered there. Likewise, the boundary of such a capsule contains all the compliant positions for the disk. Thus replacing our obstacles by such capsules translates the problem of finding a path for a disk into the simpler problem of finding a path for a point.



(a) The Minkowski sum of a disk with an obstacle resembles a capsule.

(b) The boundary $\partial U(r_o)$ of the union of all capsules of radius r_o contains exactly all positions where the object is compliant.

Figure 2.5: Computing the Minkowski sums of disks with our obstacles allows us to find all compliant positions.

That these capsules form a collection of pseudodisks follows from our obstacles being non-intersecting [15, 16]. Their union $U(r) = \bigcup_{\gamma\in\Gamma}(\gamma \oplus D(r))$ therefore has $O(n)$ complexity. Specifically, $\partial U(r)$ consists of $O(n)$ line segments and circular arcs of radius r, sharing $O(n)$ endpoints.

Figure 2.5(b) illustrates how $\partial U(r_o)$ forms all compliant positions for the object, and that this shape can be partially inside the areas blocked off from the object and pusher by obstacles. We will forgo drawing those interior parts of $\partial U(r)$ from here on, as they are not useful for path-finding.

2.4 Nieuwenhuisen's subroutine

As discussed in the introduction, Nieuwenhuisen [15, Chapter 9] presented a probabilistic algorithm for computing push plans between a given initial and destination position for the object. His algorithm repeatedly supplies a subroutine with query paths τ, asking it to produce a push plan that pushes the object along τ as far as possible (either until the end of the path, or until the object and/or pusher get stuck). Creating such a subroutine is the goal of this thesis, but Nieuwenhuisen also developed one of his own [16, 15, Chapter 8]. This section discusses Nieuwenhuisen's approach, and the rest of this thesis describes our new method and how it improves upon Nieuwenhuisen's. For understanding our method the reader is free to skip this section, and we'll only refer back to it when comparing our results with Nieuwenhuisen's.

2.4.1 Four cases

Nieuwenhuisen's method computes a contact-preserving push plan for a given query path τ by splitting the path up into k separate *sections*. It then creates push plans for each of the sections and connects those together. Three kinds of path sections can occur, and one way of connecting them:

- In a *straight-line compliant section* the object slides along the edge of an obstacle, which we'll call the *compliant obstacle*.
- In a *circular compliant section* the object rotates around a vertex of an obstacle, which we'll call the *compliant vertex*.
- In a *non-compliant section* the object moves through the work space without touching any obstacles.

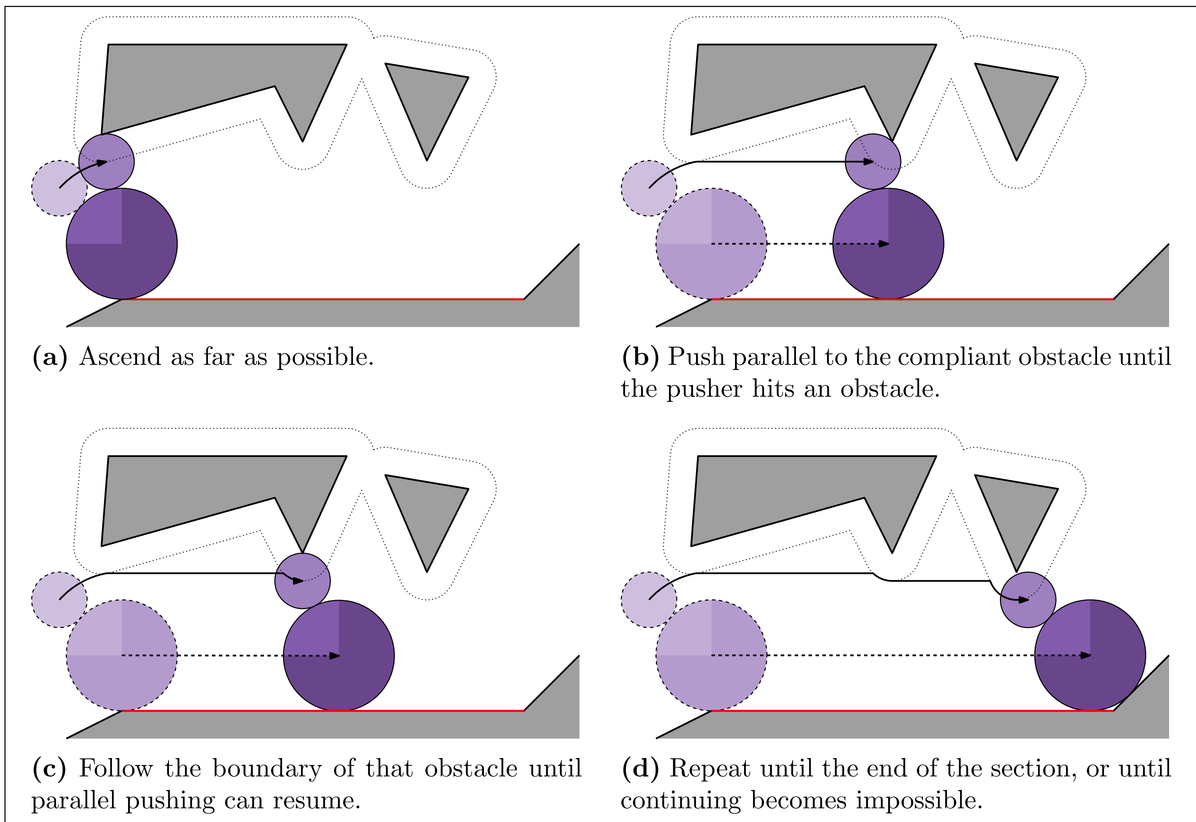
  Nieuwenhuisen's algorithm for the main problem will only produce two types of non-compliant sections in its query paths: straight lines, and hockey stick curves. His subroutine only deals with *straight-line non-compliant sections*, and the hockey stick curves are treated completely separately with numerical methods [15, Subsection 9.4.2]. The algorithm then combines these results afterwards.
- In a *contact transit* the object doesn't move at all, but the pusher rotates around the object. This is needed between sections whenever the pushing angle at the end of a section is not in the push range of the next section.

Constructing push plans for each of these four cases is discussed in the following subsections.

In all cases it is assumed that the given path doesn't move the object into any obstacles, which can be accomplished by cutting the object's path τ off at the first intersection. (This is not a restriction, because we only wanted a push plan that pushed the object as far along τ as possible.) Nieuwenhuisen does this using the same techniques discussed in Subsections 2.4.6 and 2.4.7 for the pusher's path.

2.4.2 Straight-line compliant sections

For a straight-line compliant section, the pusher is said to *descend* when it moves closer to the compliant obstacle, and to *ascend* when it moves farther away. To compute a push plan for such a section, Nieuwenhuisen's method makes the pusher do the following:



(a) Ascend as far as possible.

(b) Push parallel to the compliant obstacle until the pusher hits an obstacle.

(c) Follow the boundary of that obstacle until parallel pushing can resume.

(d) Repeat until the end of the section, or until continuing becomes impossible.

Figure 2.6: An example showing how Nieuwenhuisen's method creates push plans for straight-line compliant sections.

Step 1: Ascend as far as possible; either until the pusher hits some other obstacle, or until it is at the end of the push range. (See Figure 2.6(a).)

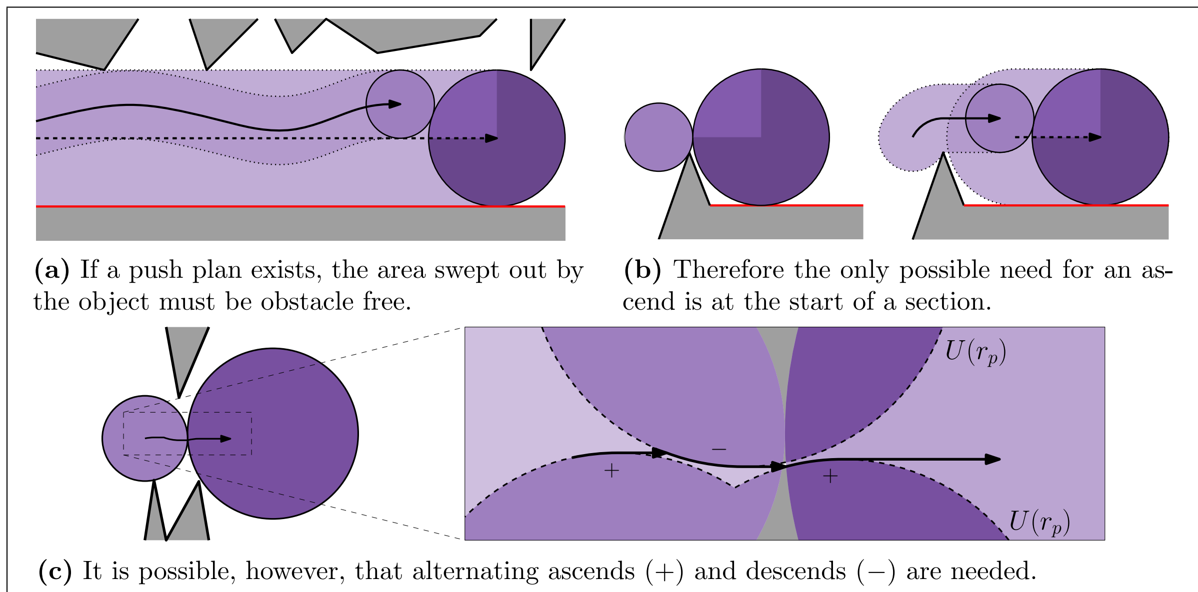
Step 2: Push parallel to the compliant obstacle until the pusher hits some other obstacle. (See Figure 2.6(b).)

Step 3: Follow the boundary of the obstacle that was hit as long as it is descending so at the end the pusher can again push parallel to the compliant obstacle. (See Figure 2.6(c).)

Step 4: Repeat steps 2 and 3 until the end of the section is reached and the push plan for this section is complete, or it becomes impossible to continue further. (See Figure 2.6(d).)

Nieuwenhuisen argues that this method will always find a contact-preserving push plan if one exists. Consider the areas swept out by the moving disks. If a push plan exists, that means the object's sweep area must be free of obstacles, and the pusher won't hit any obstacles if it moves through this same area (by staying "behind" the object, as in Figure 2.7(a)). Leaving this area by an ascend is therefore generally not needed.

However, the pusher will not have entered this area yet at the start of the section. Obstacles encountered there may call for the need to ascend (as in Figure 2.7(b)). Thus by ascending as far as possible at the start of the section, Nieuwenhuisen argues, it is then never needed to ascend again, and the procedure only fails if no contact-preserving push plan exists. However, Figure 2.7(c) shows a counter-example where alternating ascends and descends are needed. This requires one to modify step 3 to follow the boundary as long as it is descending *or ascending*.



(a) If a push plan exists, the area swept out by the object must be obstacle free.

(b) Therefore the only possible need for an ascend is at the start of a section.

(c) It is possible, however, that alternating ascends (+) and descends (−) are needed.

Figure 2.7: The pusher only needs to ascend at the beginning of a straight-line compliant section, as for the rest of the section it can stay in the obstacle-free sweep area of the object.

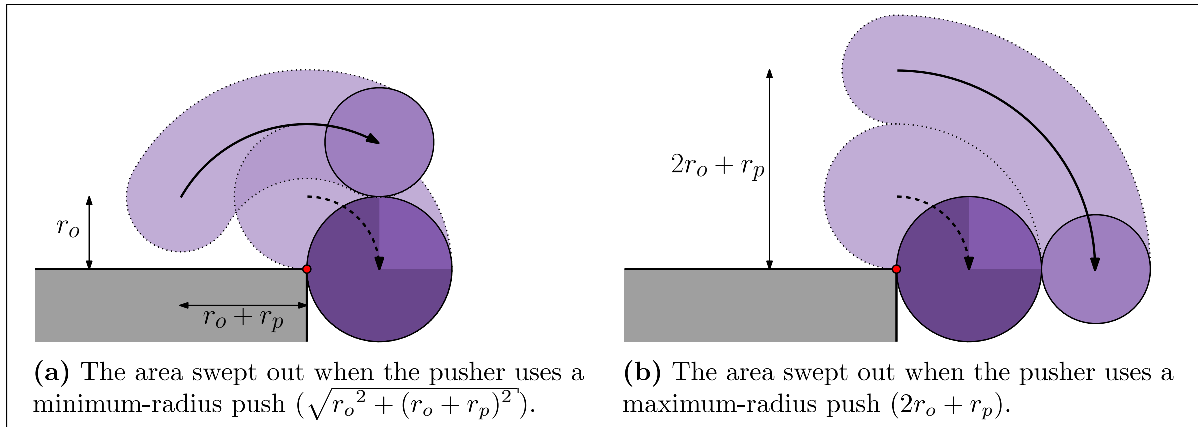
2.4.3 Circular compliant sections

Analogous to the ascending/descending terminology of the last subsection, let's call the pushing position closest to the compliant vertex *lowest*, and the one farthest away *highest*.

Maintaining the lowest pushing position throughout the section, the pusher's center will move on a circular arc with radius $\sqrt{{r_o}^2 + (r_o + r_p)^2}$ (which is minimal) around the compliant vertex, as seen in Figure 2.8(a).

Maintaining the highest pushing position throughout the section, the radius is $2r_o + r_p$ (which is maximal), as seen in Figure 2.8(b). (Actually, then the pusher's center would be on the

line through the compliant vertex and the object's center, so the object can only be pushed *into* the vertex and not around it. Thus the highest pushing position is actually not defined (there is a supremum but no maximum), and neither is ascending "as far as possible" of the previous subsection. For convenience we'll allow ourselves this slight abuse of terminology.)



(a) The area swept out when the pusher uses a minimum-radius push ($\sqrt{{r_o}^2 + (r_o + r_p)^2}$).

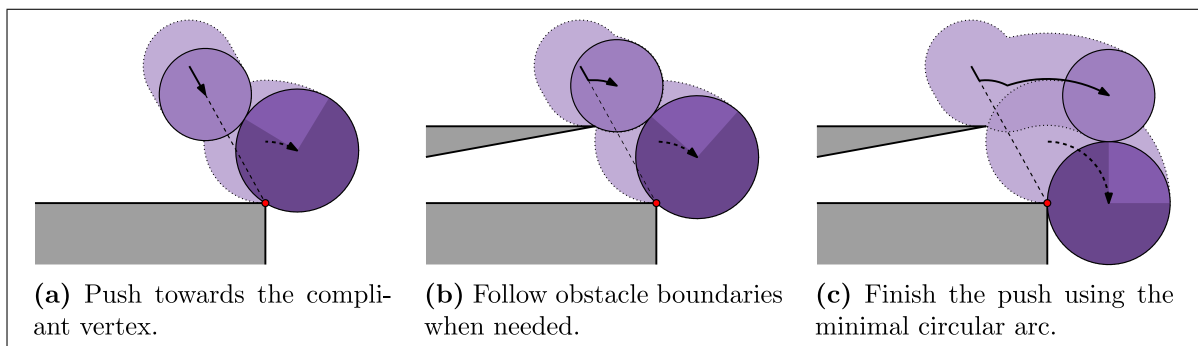
(b) The area swept out when the pusher uses a maximum-radius push ($2r_o + r_p$).

Figure 2.8: No push plan for a circular compliant section can keep the pusher in the object's sweep area. Shown are the sweep areas for the lowest and highest pushing positions.

The minimum-radius push of Figure 2.8(a) also maximizes the overlap between the two sweep areas, showing that we can't keep the pusher in the object's sweep area to avoid obstacles. What can be done, however, is trying to minimize the combined area swept out by the two disks. This is what Nieuwenhuisen's method does, as follows:

Step 1: Push in a straight line towards the compliant vertex until the lowest pushing position is reached. (See Figure 2.9(a).)
Whenever an obstacle is hit, follow obstacle boundaries until either pushing to the compliant vertex can resume, or the lowest pushing position is reached. (See Figure 2.9(b).)

Step 2: Push on the circular arc with radius $\sqrt{{r_o}^2 + (r_o + r_p)^2}$ around the compliant vertex. (See Figure 2.9(c).)
As before, when an obstacle is hit we follow obstacle boundaries until the lowest pushing position is reached (again).



(a) Push towards the compliant vertex.

(b) Follow obstacle boundaries when needed.

(c) Finish the push using the minimal circular arc.

Figure 2.9: An example showing how Nieuwenhuisen's method creates push plans for circular compliant sections.

This method maximizes the overlap between the areas swept out by the object and the pusher, thus minimizing the combined sweep area. Furthermore, every other push plan will have a combined sweep area that is a superset of this one. Thus this method finds a contact-preserving push plan if and only if one exists.

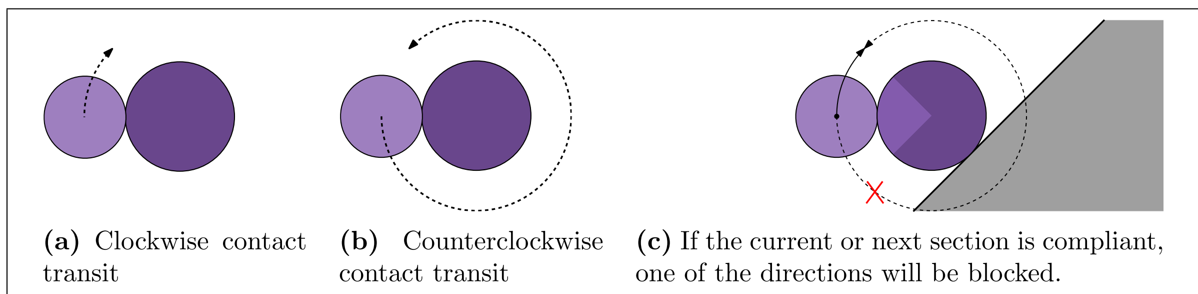
2.4.4 Straight-line non-compliant sections

As discussed in Section 2.1, and illustrated in Figure 2.1(b), there is only one possible push plan for a (straight-line) non-compliant section. If any obstacles are in the way, no push plan exists, for they cannot be avoided while still pushing the object on the correct path.

Note that for most of the straight-line non-compliant section, the pusher will stay in the sweep area of the object. Thus any obstacles that the pusher runs into (but the object doesn't) must be at the start of the section (as in Figure 2.1(b).)

2.4.5 Contact transits

A contact transit is needed whenever the push angle at the end of a section is not in the push range of the next section. It can be done either by a clockwise or a counterclockwise turn around the object (see Figure 2.10(a)–(b)).



(a) Clockwise contact transit **(b)** Counterclockwise contact transit **(c)** If the current or next section is compliant, one of the directions will be blocked.

Figure 2.10: Contact transits can be done either clockwise or counterclockwise, but one (or both) of these may be blocked by obstacles.

When neither of these arcs is blocked, the pusher can just pick one (say, the shortest). When one is blocked, for example when the current or next section is compliant, there is no choice (see Figure 2.10(c)). When both are blocked, no contact-preserving push plan exists.

2.4.6 Ray shooting queries

To implement the algorithms for straight-line compliant and non-compliant sections, a way is needed to check whether and where the pusher hits obstacles when moving on straight-line paths. The first half of the algorithm for circular compliant sections also needs this. Nieuwenhuisen does this by computing the first intersection of rays (i.e. half-lines) with $\partial U(r_p)$, using a data structure by Koltun [9]. Recall that $\partial U(r_p)$ consists of $O(n)$ line segments and circular arcs. The data structure can then be built in $O(n^2 \log n)$ time and uses $O(n^2)$ space, after which each ray can be processed in $O(\log n)$ time.

We'll denote the number of sections per type as k_s (straight-line compliant), k_c (circular compliant), and k_n (straight-line non-compliant), such that $k = k_s + k_c + k_n$. Then the following numbers of these "ray shooting queries" need to be performed:

- **Straight-line compliant sections**
  No obstacle can be hit more than once per section, but in the worst case (a constant fraction of) all the obstacles will be hit by the pusher at every straight-line compliant section. Thus $O(k_s n)$ ray shooting queries may be needed.

- **Circular compliant sections**
  On the straight-line pushes towards the compliant vertex, again (a constant fraction of) all the obstacles could be hit by the pusher at every circular compliant section. Thus $O(k_c n)$ ray shooting queries need to be performed.

- **Non-compliant sections**
  There is only one possible push plan for a straight-line non-compliant section, and that is itself a straight line. Thus $O(k_n)$ ray shooting queries are needed.

This gives a total of $O(k_s n + k_c n + k_n)$ ray shooting queries. An upper bound for this is $O(kn)$, which is tight if there are no non-compliant sections. Thus the total time needed for ray shooting queries is $O(kn \log n)$ per path τ, after a preprocessing time of $O(n^2 \log n)$. The space used is $O(n^2)$.

2.4.7 Circular arc queries

For contact transits (including the ascends at the start of straight-line compliant sections) a way is needed to check whether and where the pusher hits obstacles when moving on circular arcs. The second half of the algorithm for circular compliant sections also needs this. For this, Nieuwenhuisen uses an algorithm by Balaban [3]. Instead of doing the queries one arc at a time, multiple arcs are processed together. If m is the number of arcs to be tested, then the algorithm uses $O((n+m)\log(n+m)+q)$ time and $O(n+m)$ space to output all q intersections of the m arcs with $\partial U(r_p)$.

The following numbers of arcs need to be intersected:

- **Circular compliant sections**
  Nieuwenhuisen doesn't compute the intersections of these circular arcs whenever a path τ is given and the circular compliant sections become known. Instead, he computes the intersections of *all* potential circular arcs that could occur in circular compliant sections of given paths. This is done by placing a full circle of radius $\sqrt{{r_o}^2 + (r_o + r_p)^2}$ on each of the $O(n)$ obstacle vertices, and intersecting these with $\partial U(r_p)$.

  In the worst case, when obstacles are tightly clustered, each of these $m = O(n)$ arcs could intersect $O(n)$ obstacles, leading to $q = O(n^2)$ intersections. Thus it takes $O(n^2)$ time and $O(n)$ space to compute these intersections, but storing them takes $O(n^2)$ space. Whenever a new path τ is given for the same environment, the $O(kn)$ relevant intersections can be looked up and processed in $O(kn)$ time.

- **Contact transits**
  There is (potentially) a contact transit at the start of each of the sections, thus there are $m = O(k)$ contact transits. All of these are circular arcs with radius $r_o + r_p$. In the worst case, (a constant fraction of) all obstacles are encountered at the start of every section, leading to $q = O(kn)$ intersections. Thus all possible intersections from contact transits can be found in $O((k + n)\log(k + n) + kn)$ time, and $O(k + n)$ space.

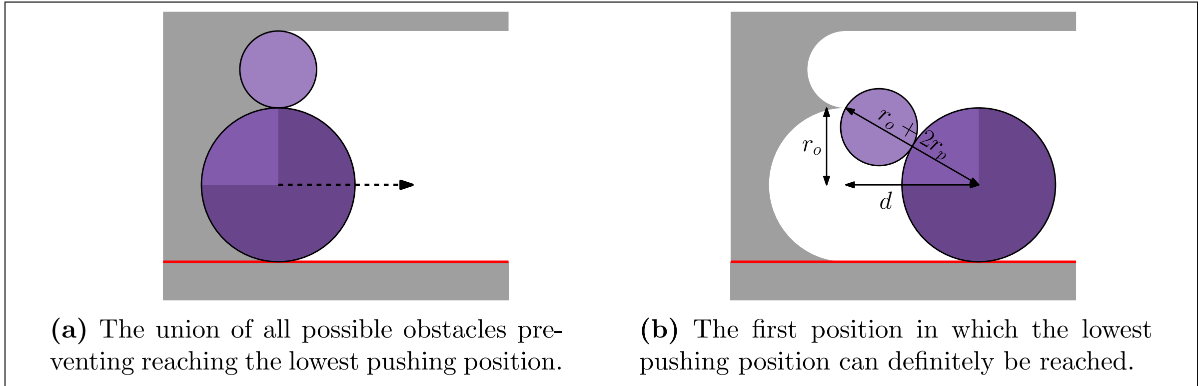
Thus the total time needed for these "circular arc queries" is $O((k+n)\log(k+n)+kn)$, with a preprocessing time of $O(n^2)$. The space used is $O(n^2 + k)$.

Combining the time for the two types of queries, Nieuwenhuisen's subroutine needs $O(kn \log n)$ time, after a preprocessing time of $O(n^2 \log(n))$. The total space usage is $O(n^2 + k)$. The computed push plan will consist of $O(kn)$ sections.

2.4.8 Low obstacle density

Worst cases with very densely packed obstacles don't tend to occur much in practice. Thus it makes sense to study the planning of paths in spaces with *low obstacle density* [4], as defined next.

Recall from Subsection 2.4.2 (especially Figure 2.7) that the pusher is safe from obstacles if it stays in the object's sweep area, but that obstacles at the start of the sweep area can prevent it from getting there right away. Figure 2.11(a) shows the union of all such possible obstacles. After the object has moved a distance $d = \sqrt{(r_o + 2r_p)^2 - r_o{}^2} = \sqrt{4r_p{}^2 + 4r_p r_o}$, those obstacles can't possibly interfere with the pusher getting in the object's sweep area anymore, as shown in Figure 2.11(b).



(a) The union of all possible obstacles preventing reaching the lowest pushing position.

(b) The first position in which the lowest pushing position can definitely be reached.

Figure 2.11: The minimum distance d that the object has to move on a straight-line compliant section to guarantee the pusher can reach the lowest pushing position.

So after that distance d we're in the clear, but in this initial distance there could be very many obstacles. Realistically this won't happen though, as we can often assume that:

- The environment has a *low obstacle density* [4], defined as follows:

  The environment is a λ-low-density environment when for any disk D of diameter x the number of obstacles $\gamma \in \Gamma$ with length$(\gamma) \geq x$ that intersect D is at most λ. The density of the environment is the smallest λ for which it is a λ-low-density environment. We say the environment has *low obstacle density* if its density is a small constant.

- The radius of the object is at most a constant times the length of the smallest obstacle. More specifically, that $4r_o < c\sigma$ holds for some (small) constant $c > 0$, with σ being the length of the smallest obstacle.

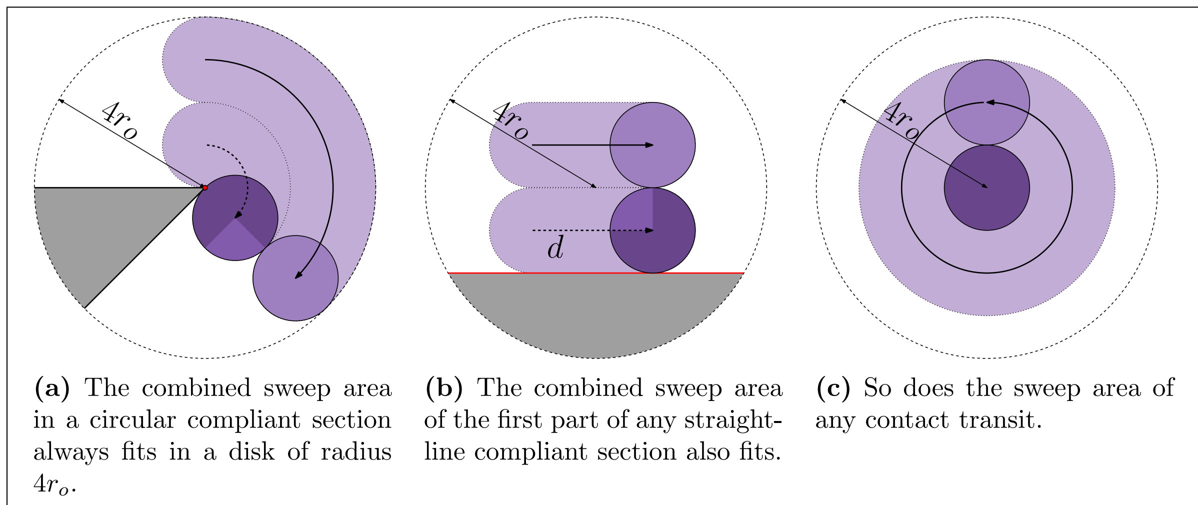
Together, these assumptions imply that any disk with radius $4r_o$ will intersect only a constant number of obstacles. We will show that the combined sweep areas of the object and pusher for a path section will always fit into such a disk, yielding a push plan complexity of $O(k)$ instead of $O(kn)$.

Recall that we assumed the pusher to be smaller than the object, i.e. $r_p < r_o$. Since bigger disks sweep out more area, we'll consider the limiting case of $r_p = r_o$ to get an upper bound for the size of the sweep areas.

In a circular compliant the section the combined sweep area is largest when the highest pushing position is maintained. That this fits in a disk with radius $4r_o$ even when $r_p = r_o$ is illustrated in Figure 2.12(a).

In a straight-line compliant the section the combined sweep area is also largest when the highest pushing position is maintained. For the initial push of length d this area also fits in a disk with radius $4r_o$, as seen in Figure 2.12(b). After this distance the pusher can continue in the object's sweep area and won't need to encounter any more obstacles. (Note, however, that Nieuwenhuisen's approach described in Subsection 2.4.2 can still encounter $\Theta(n)$ obstacles after the initial push of length d, as it only descends the pusher as necessary rather than moving into the object's sweep area as soon as possible. So again his approach needs a slight modification before these claims hold.)

In a contact transit the largest sweep area occurs when the transit is a (nearly) full circle. That this area also fits in a disk with radius $4r_o$ is illustrated in Figure 2.12(c).



(a) The combined sweep area in a circular compliant section always fits in a disk of radius $4r_o$.

(b) The combined sweep area of the first part of any straight-line compliant section also fits.

(c) So does the sweep area of any contact transit.

Figure 2.12: The combined sweep areas of compliant sections and contact transits always fit in a disk of radius $4r_o$.

Thus if we assume low obstacle density, and that the radius of the object is at most a constant times the size of any obstacle, then in every section at most $O(1)$ obstacles are encountered (assuming the procedure of Subsection 2.4.2 is modified to make the pusher descend into the

object's sweep area when it can). This brings the push plan complexity down to $O(k)$, and the running time to $O((k+n)\log(k+n))$. (Nieuwenhuisen claims the new running time is $O((k+n)\log n)$, most likely an oversight.) The preprocessing time of $O(n^2 \log n)$ and space usage of $O(n^2 + k)$ remain the same.

Chapter 3

Pushing while maintaining contact

A general-purpose technique for traditional path-planning is to translate the problem from the *work space* into the *configuration space*. The work space is the given environment in which the robot needs to find a path, and a *configuration* is one specific placement of the robot. Each point in the configuration space corresponds to such a configuration in the work space. For example, a rectangular robot moving around in a 2-dimensional work space has a 3-dimensional configuration space, with two coordinates representing the robot's location, and one its orientation. Some configurations will be invalid, for example because the robot intersects one of the obstacles, and such points form the *forbidden (configuration) space*. The remaining points form the *free (configuration) space*, representing all the valid configurations. A path through the free space from the initial configuration to a destination configuration then immediately translates back to a solution of the original problem in the work space.

Our subroutine for finding contact-preserving push plans given an object path τ is based on this general-purpose technique: we first compute the configuration space and then find a path through the free space from an initial configuration to a destination configuration, yielding a push plan. The shape of the configuration space for this problem is not as straightforward as with traditional translational path planning, though. Not only does a configuration correspond to a placement of both the object *and* the pusher in the work space, but configurations can be invalid even when there are no intersections with obstacles but the object simply isn't being pushed along path τ. The latter also has implications for the path we eventually need to find through the free space.

To manage this complexity we impose several restrictions on the given object path τ. In particular, we require that all path sections are "well-behaved" in some sense, as described in Section 3.1. This still yields a more general notion of path sections than Nieuwenhuisen's, though, and we'll show that the path-section types he distinguishes are all well-behaved.

In Section 3.2 we discuss the shape of the resulting configuration space and derive its complexity, and Section 3.3 then gives an algorithm to compute it from the given work space.

Sections 3.4 and 3.5 discuss the problem of finding a path through the free space that will result in a valid contact-preserving push plan.

Finally, Section 3.6 describes how the low-obstacle-density assumptions described in Subsection 2.4.8 yield lower worst-case running times for our subroutine just as they did for Nieuwenhuisen's.

3.1 Well-behaved path sections

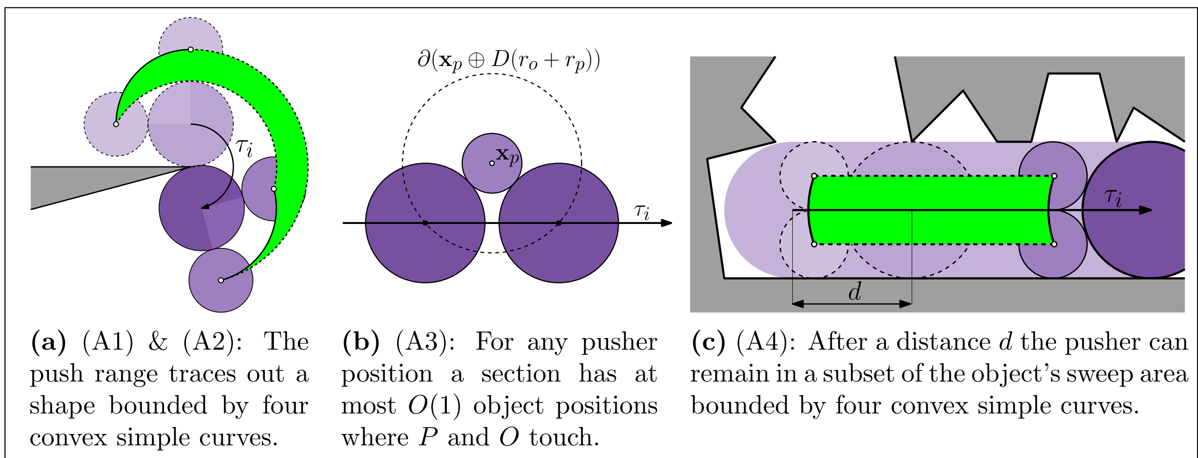
We consider a collection of curves to be *simple curves* when each curve is C^0-continuous, we can compute its length and $O(1)$ points of intersection with a circle or line in $O(1)$ time, and we can compute the intersections of any two such curves in $O(1)$ time. We then require our given object path τ to consist of a sequence of k *well-behaved path sections* $\tau_1, \tau_2, \ldots, \tau_k$, which are simple curves connected end-to-end. More precisely, let path τ be parametrized with a parameter $s \in [0, 1]$, then each section τ_i has s in some interval $[b_i, e_i] \subseteq [0, 1]$ such that $\tau(s) = \tau_i(s)$ for all $s \in [b_i, e_i]$ and $b_1 = 0$, $e_k = 1$, $e_i = b_{i+1}$ for $1 \leq i < k$. In addition, each well-behaved path section satisfies the following properties for pushing the object along it:

(A1) At any position $\tau_i(s)$ of the object along section τ_i, the positions for the pusher that allow it to push the object further along τ form a single continuous range (the push range) that can be computed in $O(1)$ time. The two endpoints of the push range will be referred to as the *lowest* and *highest* pushing positions such that the push range is a counterclockwise arc from the lowest to the highest pushing position.

(A2) As the object moves along section τ_i, the push range traces out a connected shape without holes bounded by four convex simple curves: the push range at the start and end of τ_i, and the paths traced out by the lowest and highest pushing positions. (See Figure 3.1(a).) Furthermore, we can compute these curves from τ_i in $O(1)$ time.

(A3) For any pusher position there are at most $O(1)$ object positions in a section where the pusher touches the object. Note that this follows directly from τ_i being a simple curve as these $O(1)$ object positions for an arbitrary pusher position $\mathbf{x}_p$ are given by $\tau_i \cap \partial(\mathbf{x}_p \oplus D(r_o + r_p))$. (See Figure 3.1(b).)

(A4) After the object has moved a certain distance $d = O(r_o)$ 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. The paths traced out by lowest and highest pushing positions in this sweep area are convex simple curves that can be computed in $O(1)$ time. (See Figure 3.1(c).)



(a) (A1) & (A2): The push range traces out a shape bounded by four convex simple curves.

(b) (A3): For any pusher position a section has at most $O(1)$ object positions where P and O touch.

(c) (A4): After a distance d the pusher can remain in a subset of the object's sweep area bounded by four convex simple curves.

Figure 3.1: The four well-behavedness properties that we require of the given path sections.

Property (A4) isn't needed for correctness, but as we'll see in Section 3.6 it will allow for better time bounds for our subroutine when applied to low-obstacle-density environments. These four properties together capture all of the path-section types supported by Nieuwenhuisen's subroutine under a single notion:

Theorem 3.1. *Nieuwenhuisen's straight-line compliant, circular compliant, and straight-line non-compliant path sections are all well-behaved.*

Proof. Line segments and circular arcs are of course simple curves, and property (A3) then trivially holds. Property (A1) holds for Nieuwenhuisen's path sections by definition, so it remains to show that properties (A2) and (A4) hold:

(A2) The push range in Nieuwenhuisen's path sections is of fixed size throughout the section and merely turns along with τ_i. Thus a connected shape without holes is traced out that is bounded by the push ranges at the start and end of the section, and the paths of the lowest and highest pushing position.

For both types of straight-line sections the lowest and highest pushing positions trace out line segments that are simply displaced versions of the path section itself. For circular compliant sections the lowest and highest pushing position trace out circular arcs of radius $\sqrt{{r_o}^2 + (r_o + r_p)^2}$ and $2r_o + r_p$, respectively, as mentioned in Subsection 2.4.3. (See also Figure 3.1(a).)

(A4) A circular compliant section always has a length of less than $2\pi r_o$ so this property trivially holds. In a straight-line non-compliant section the pusher will be completely in the sweep area of the object after a distance $2r_p$, and then the whole push range will be in that sweep area. In a straight-line compliant section the pusher can remain in the object's sweep area after a distance $\sqrt{4{r_p}^2 + 4r_p r_o}$ (see Subsection 2.4.8), and the area where it does so is bounded by two circle arcs and two line segments (see Figure 3.1(c)). □

It is even the case that hockey-stick non-compliant sections are "almost" well-behaved. The exact intersections between a hockey-stick curve and a circle or line can't be computed in $O(1)$ time, but the curve can be approximated fairly well by a small number of quadratic Bézier curves which do allow this. The result is a well-behaved path section:

Theorem 3.2. *Nieuwenhuisen's hockey-stick non-compliant sections become well-behaved path sections when approximated by a constant number of quadratic Bézier curves.*

Proof. Quadratic Bézier curves are simple curves, and property (A3) then trivially holds. Property (A1) holds for Nieuwenhuisen's path sections by definition, so it remains to show that properties (A2) and (A4) hold:

(A2) In a non-compliant section the push range consists of a single angle at all times so the push range traces out a single curve, which is then the path traced out by both the lowest and highest pushing position. For a hockey-stick non-compliant section τ_i this path is a line segment as discussed in Section 2.2.

(A4) The hockey-stick non-compliant sections generated by Nieuwenhuisen's global algorithm are all of the same shape, just of different orientations. Their length then depends on r_p and r_o only, making it an $O(r_o)$ constant. (Note that this value grows to infinity as r_p approaches r_o, though.) □

3.2 Shape of the configuration space

Before trying to compute the configuration space we'll discuss its shape and complexity. Recall that a configuration is a placement of the object and the pusher in the work space. Thus we can view it as a pair of positions $(\mathbf{x}_o, \mathbf{x}_p) \in \mathbb{R}^2 \times \mathbb{R}^2$. However, this gives four *degrees of freedom* where there are really only two. We assumed the object path $\tau : [0, 1] \to \mathbb{R}^2$ is given, and (for now) that the pusher has to maintain contact with the object at all times, so a configuration can be represented as a pair $(s, \theta) \in [0, 1] \times \mathcal{S}^1$. Here s is the fraction of the path already traversed by the object (i.e. $\mathbf{x}_o = \tau(s)$), and θ is the pushing angle; the angle that the line from the pusher's center to the object's center makes with the positive x-axis.

The configuration space is the parameter space of these configurations, and as such is a 2-dimensional space with the topology of a cylinder. In this space an obstacle γ is no longer simply a line segment. Since each point represents a configuration, i.e. a certain placement of both the pusher and the object in the work space, a *configuration-space obstacle* $\mathcal{C}_\gamma$ consists of the configurations where the pusher intersects that obstacle γ. (Recall that the object never intersects any obstacles along path τ.) The open boundary of $\mathcal{C}_\gamma$ then consists of the configurations where the pusher is compliant with γ.

Figure 3.2(a) shows an example of a work space, with the corresponding configuration space cylinder in Figure 3.2(b) showing the union of configuration-space obstacles. For clarity, we'll not show such cylinders anymore from here on, and will instead visualize configuration spaces in a "flattened" way, as in Figure 3.2(c), with the s-axis drawn horizontally and the θ-axis drawn vertically.

Apart from the pusher intersecting any of the obstacles, a configuration can also be forbidden because it has the pusher outside of the push range for that position of the object along path τ. We'll call these configurations the *forbidden push ranges* (drawn in dark gray), and their union with the configuration-space obstacles (drawn in light gray) forms the full forbidden space. (Figures 3.4(a), 3.7(b), and 4.9(b) further on show examples.)

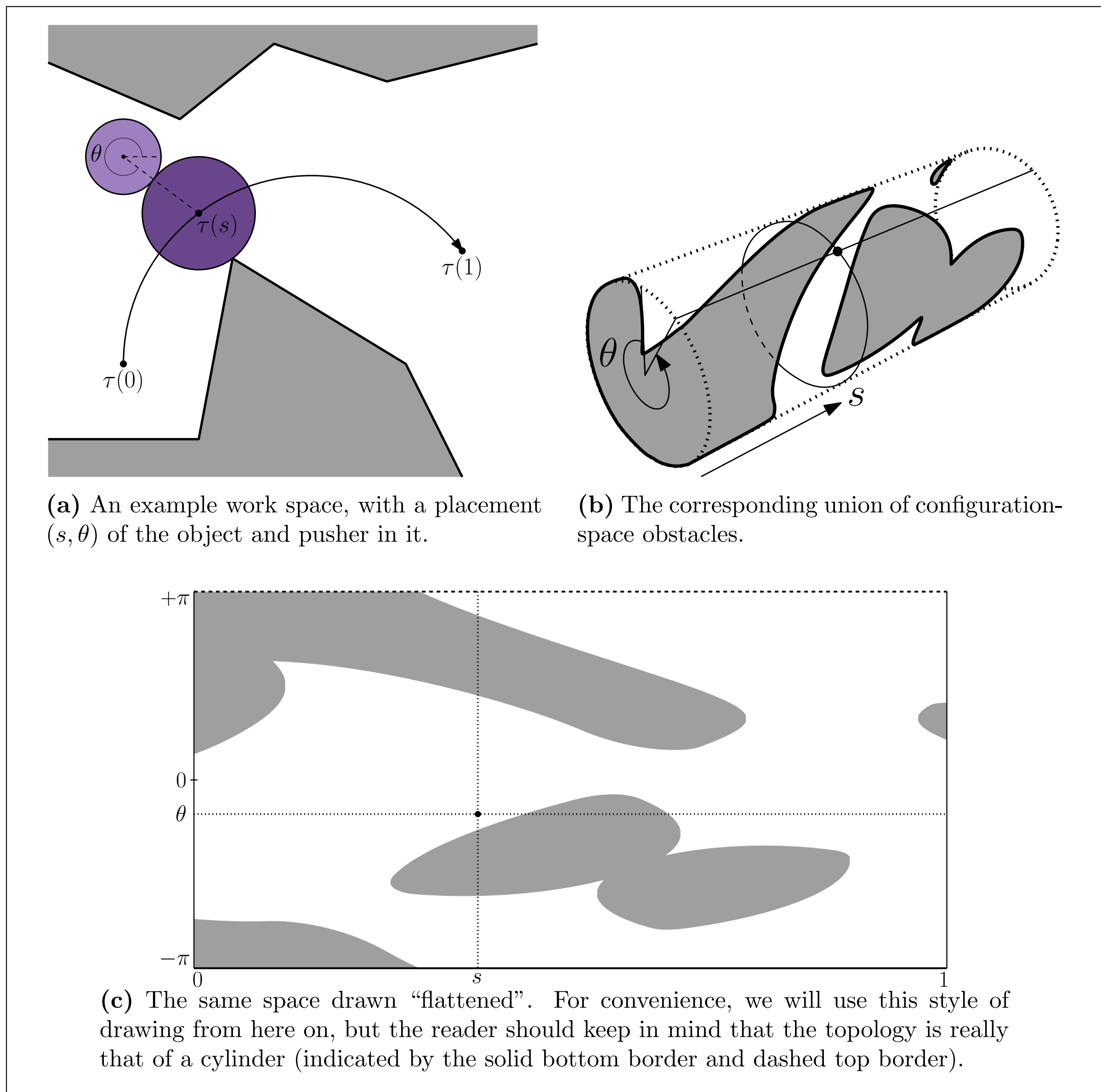
The shape of the configuration-space obstacles, the forbidden push ranges, and their union will be discussed over the next few subsections. For now we're not interested so much in the form of the boundary curves of these shapes, but more in the number of vertices of those curves. This number gives us the combinatorial complexity of the configuration space, which gives an indication of the (potential) efficiency of our approach. We state this complexity now, and prove it through a series of lemmas below:

Theorem 3.3. *The configuration space has complexity $O(kn)$, i.e. the boundary of the forbidden space consists of $O(kn)$ vertices and curves between them.*

Proof. From Lemmas 3.4 through 3.7, defined and proven in the coming subsections, it follows that the complexity of the configuration space is $O(n)$ per path section. This yields a complexity of $O(kn)$ for the configuration space all k path sections. □

3.2.1 Path sections in the configuration space

As described in Section 3.1, the path sections form a partitioning of path τ. This then also induces a partitioning of the configuration space along the s-axis. For any path section



(a) An example work space, with a placement (s, θ) of the object and pusher in it.

(b) The corresponding union of configuration-space obstacles.

(c) The same space drawn "flattened". For convenience, we will use this style of drawing from here on, but the reader should keep in mind that the topology is really that of a cylinder (indicated by the solid bottom border and dashed top border).

Figure 3.2: An example work space and the corresponding union of configuration-space obstacles. The marked point in the configuration space corresponds to the configuration of the object and pusher shown in the work space.

$\tau_i : [b_i, e_i] \to \mathbb{R}^2$ we'll refer to the vertical lines $s = b_i$ and $s = e_i$ as its *section boundaries*. The area in between these two lines then contains all configurations with the object at a position along that path section. Whenever no confusion arises we'll use the term path section for both the actual section τ_i of the path and for the corresponding strip of the configuration space. Figure 3.3(b)–(c) shows an example with three path sections (and thus four section boundaries).

In the following subsections we restrict our attention to the part of the configuration space within a single path section τ_i, as it is easier to reason about.

3.2.2 A configuration-space obstacle

Recall that a configuration-space obstacle $\mathcal{C}_\gamma$ consists of the configurations where the pusher intersects γ, and that its open boundary consists of the configurations where the pusher is compliant with γ. This shape can consist of several components, both within and across sections. As just mentioned we restrict our attention to one path section τ_i, and we refer to the part of $\mathcal{C}_\gamma$ in this section as $\mathcal{C}_{\gamma,i}$. Next we show that such a shape has constant complexity:

Lemma 3.4. *The part $\mathcal{C}_{\gamma,i}$ of a configuration-space obstacle $\mathcal{C}_\gamma$ that falls within a single path section τ_i is s-monotone and consists of $O(1)$ components of $O(1)$ complexity.*

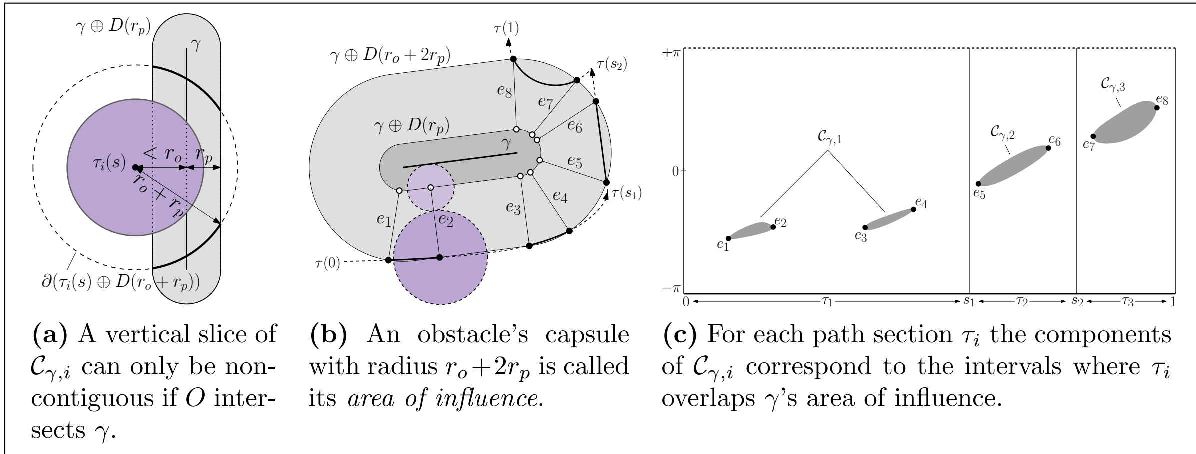
Proof. Consider the vertical slice of $\mathcal{C}_{\gamma,i}$ for the object position $\tau_i(s)$. The positions where the pusher touches the object are given by the circle $\partial(\tau_i(s) \oplus D(r_o + r_p))$. The positions where the pusher intersects obstacle γ are given by the capsule $int(\gamma \oplus D(r_p))$. The pusher positions of the vertical slice of $\mathcal{C}_{\gamma,i}$ are thus given by the intersection of this circle and capsule. Potentially, such an intersection could consists of two or more continuous ranges, but as illustrated in Figure 3.3(a) this can only happen when the object intersects obstacle γ, contradicting our assumption about path τ. Thus a vertical slice of $\mathcal{C}_{\gamma,i}$ is always a single continuous range, making $\mathcal{C}_{\gamma,i}$ an s-monotone shape.

The intersection of circle $\partial(\tau_i(s) \oplus D(r_o + r_p))$ and capsule $int(\gamma \oplus D(r_p))$ is non-empty exactly when $\tau_i(s)$ lies in the bigger capsule $int(\gamma \oplus D(r_o + 2r_p))$, which we call γ's *area of influence*. Thus the components of $\mathcal{C}_{\gamma,i}$ correspond to the intervals of τ_i that intersect γ's area of influence (see Figure 3.3(b)–(c)). Since τ_i is a simple curve, this means $\mathcal{C}_{\gamma,i}$ has at most $O(1)$ components. The shape of a component comes forth from the intersection of a capsule with a circle moving over a simple curve. All of these have $O(1)$ complexity, so the component itself does as well. □

3.2.3 Union of all configuration-space obstacles

From Lemma 3.4 it trivially follows that all n configuration-space obstacle pieces $\mathcal{C}_{\gamma,i}$ for the section together have $O(n)$ complexity. However, when we take their union new vertices may emerge as the intersections of the boundaries of different pieces. If all can intersect all others, this might yield $\Theta(n^2)$ extra vertices. Fortunately, this situation cannot actually occur:

Lemma 3.5. *The union of the n configuration-space obstacle pieces for a single path section τ_i has complexity $O(n)$.*



(a) A vertical slice of $\mathcal{C}_{\gamma,i}$ can only be non-contiguous if O intersects γ.

(b) An obstacle's capsule with radius r_o+2r_p is called its *area of influence*.

(c) For each path section τ_i the components of $\mathcal{C}_{\gamma,i}$ correspond to the intervals where τ_i overlaps γ's area of influence.

Figure 3.3: A configuration space obstacle is s-monotone and consists of $O(1)$ components per path section, each having $O(1)$ complexity.

Proof. Recall that the boundary of a configuration-space obstacle $\mathcal{C}_\gamma$ corresponds to configurations where the pusher is compliant with the obstacle γ in the work space. Thus a point of intersection of two (or more) configuration-space obstacles corresponds to the pusher being compliant with two (or more) obstacles γ_1 and γ_2. Such a pusher position is the intersection of the boundaries of the capsules $\gamma_1 \oplus D(r_p)$ and $\gamma_2 \oplus D(r_p)$. As mentioned in Section 2.3, such capsules are pseudodisks and the union of n of them therefore has $O(n)$ complexity. This means there are also only $O(n)$ positions where the pusher would be compliant with two or more obstacles.

By property (A3) for well-behaved path sections, for each of these $O(n)$ pusher positions there are at most $O(1)$ object positions in section τ_i where the pusher and object touch, hence the union has only $O(n)$ vertices that are intersections of different configuration-space obstacles. □

3.2.4 A forbidden push range

The union of configuration-space obstacles captures the configurations where the pusher intersects any of the obstacles. However, we also want to disallow configurations where the pusher isn't in a position to push the object further along its path, i.e. where the pusher is outside of the push range. We'll call the complement of the push range the *forbidden push range*. Stretching terminology a bit, we'll not only refer to the forbidden push range for a single object position, but also to the forbidden push range FPR_i for a full path section τ_i. By this we mean the set of all configurations with the object at some position along the path section and the pusher in the forbidden push range for that position. Note that FPR_i is an open shape: non-empty only within the interior of the path sections, and empty at the section boundaries. This is to allow the pusher to perform a contact transit at section boundaries, which is necessary when the push ranges of those adjacent sections don't overlap.

The asymptotic complexity of this shape is constant:

Lemma 3.6. *The forbidden push range FPR_i for a path section τ_i is s-monotone and has $O(1)$ complexity.*

Proof. From property (A1) of well-behaved path sections it follows that the forbidden push range for an object position is always a single contiguous range, bounded by a lowest and highest pushing positions. This implies that FPR_i is an s-monotone shape, whose left and right side are vertical lines along the section's boundaries.

From property (A2) it follows that this lowest and highest pushing position trace out two non-crossing simple curves as the object moves along the simple curve τ_i. Thus the lower and upper boundary of FPR_i are also continuous, non-crossing, $O(1)$-complexity curves □

3.2.5 The full forbidden space

With FPR_i so defined, the full forbidden space for section τ_i is the union of FPR_i and $\bigcup_{\gamma\in\Gamma}\mathcal{C}_{\gamma,i}$. The latter has complexity $O(n)$ and the former $O(1)$, but their union can have additional vertices from intersections between their respective boundaries. Again, there are never more than $O(n)$ such intersections:

Lemma 3.7. *In a path section τ_i the boundaries of the forbidden push range FPR_i and the union of configuration-space obstacle pieces $\bigcup_{\gamma\in\Gamma}\mathcal{C}_{\gamma,i}$ intersect in at most $O(n)$ points.*

Proof. The boundary of $\bigcup_{\gamma\in\Gamma}\mathcal{C}_{\gamma,i}$ corresponds to configurations where the pusher is complaint with any of the obstacles. Such pusher positions are given by $\partial U(r_p)$, a shape of $O(n)$ line segments and circular arcs. The pusher positions for the lower and upper boundaries of FPR_i are given by simple curves by property (A2) of well-behaved path sections. The left and right boundaries of FPR_i are along the section boundaries and their pusher positions are thus given by circular arcs. These four curves can all intersect $\partial U(r_p)$ at most $O(n)$ times, which yields the $O(n)$ positions where the pusher is both compliant with some obstacle and on the edge of the forbidden push range for the section. By property (A3) each of these pusher positions occurs in at most $O(1)$ configurations thus FPR_i and $\bigcup_{\gamma\in\Gamma}\mathcal{C}_{\gamma,i}$ have at most $O(n)$ intersections. □

Thus the forbidden space for a single path section has complexity $O(n)$, for a total of $O(kn)$ over the whole path τ as claimed.

3.3 Computing the configuration space

Having established the parts that make up the shape of the forbidden space, our algorithm for computing the configuration space becomes a fairly straightforward four-step process. For each of the k path sections τ_i:

1. Compute the n configuration-space obstacles $\mathcal{C}_{\gamma,i}$.
2. Compute the forbidden push range FPR_i.
3. Compute the union of these shapes to get the full forbidden space.
4. Compute a vertical decomposition of the free space.

Each of these steps will be described in detail in the following subsections.

3.3.1 Representing configuration-space curves

To compute the configuration space for a path section τ_i we need a way to work with the curves bounding its forbidden push range and configuration-space obstacles. The type of direct equations of these curves in general don't allow intersection computations to be performed exactly. Rather than resorting to numerical approximation, we'll represent the specific curves of our problem in a more indirect way that does allow exact computation.

With each configuration-space curve C in section τ_i we associate a simple curve C_o of object positions, and a simple curve C_p of pusher positions. We choose these curves to contain (a superset of) all the configurations of C, and to have the property that from a given point $\mathbf{x}_p \in C_p$ we can compute in $O(1)$ time with which points on C_o it forms configurations of C. This is accomplished as follows:

- When C is a vertical line segment we call it a *V-curve* (V for "vertical").

  In this case we take C_o to be the single object position $\tau_i(s)$ shared by all configurations of C, and C_p to be the arc on the circle $\partial(\tau_i(s) \oplus D(r_o + r_p))$ given by the pusher positions of C. There is then a trivial bijection between C_p and C.

- When C is (part of) a lower boundary (either of FPR_i, or of a component of some $\mathcal{C}_{\gamma,i}$) we call it a *lower H-curve* (H for "horizontal").

  In this case we take C_o to be the part of τ_i given by the object positions of C. When C is part of FPR_i we take C_p to be the simple curve traced out by the lowest pushing position along the section, as per property (A2). When C is part of $\mathcal{C}_{\gamma,i}$ we take $C_p = \partial(\gamma \oplus D(r_p))$.

  A pusher position $\mathbf{x}_p \in C_p$ forms configurations where the object and pusher touch with the set of object positions $S_o = C_o \cap \partial(\mathbf{x}_p \oplus D(r_o + r_p))$, but not all of these configurations need to be part of C. An object position $\mathbf{x}_o \in S_o$ similarly forms configurations with the set of pusher positions $S_p = C_p \cap \partial(\mathbf{x}_o \oplus D(r_o + r_p))$. Then $\mathbf{x}_p$ forms configurations of C exactly with those $\mathbf{x}_o \in S_o$ for which $\mathbf{x}_p$ is the lowest pushing position in S_p. (This calculation can be done in $O(1)$ time because both sets have size $O(1)$ by being the intersections of a circle and a simple curve.)

- When C is (part of) an upper boundary (either of FPR_i, or of a component of some $\mathcal{C}_{\gamma,i}$) we call it an *upper H-curve*, and define it analogously to a lower H-curve.

With this representation, the intersections between two curves C_1 and C_2 can be computed by taking the intersection of $C_{1,p}$ and $C_{2,p}$ and finding which corresponding points on $C_{1,o}$ and $C_{2,o}$ it forms configurations of C_1 and C_2 with. Splitting such curves C at an intersection point can be done by splitting C_p for V-curves or C_o for H-curves.

3.3.2 Computing a configuration-space obstacle

The part $\mathcal{C}_{\gamma,i}$ of a single configuration-space obstacle $\mathcal{C}_\gamma$ for a single path section τ_i consists of $O(1)$ components by Lemma 3.4. As each component is s-monotone by the same lemma, its boundary consists of a lower and an upper H-curve. The C_p parts of these are all simply

$\partial(\gamma \oplus D(r_p))$, and as mentioned in the proof of Lemma 3.4 we can find the C_o parts as the intervals of τ_i intersecting γ's area of influence, i.e. the capsule $int(\gamma \oplus D(r_o + 2r_p))$. When a component's lower and upper H-curve touch a section boundary we'll need to add a V-curve between these endpoints to produce a closed shape.

Thus we can compute the (at most four) curves bounding each of the $O(1)$ components of $\mathcal{C}_{\gamma,i}$ in $O(1)$ time, for a total of $O(kn)$ time over all n obstacles γ and all k path sections τ_i.

3.3.3 Computing the forbidden push range

Like a component of $\mathcal{C}_{\gamma,i}$, the forbidden push range FPR_i for a single path section τ_i is also an s-monotone shape bounded by (at most) four curves, by Lemma 3.6. The C_p parts of the lower and upper H-curve can be computed in $O(1)$ time by property (A2), and their C_o parts are simply τ_i. At both section boundaries we connect the endpoints of the lower and upper H-curve by a V-curve to close the shape.

Thus we can compute FPR_i in $O(1)$ time, for a total of $O(k)$ time over all k path sections τ_i.

3.3.4 Computing the forbidden space union

The full forbidden space for section τ_i is the union of the configuration-space obstacles and the forbidden push range. We can compute the union of these $O(n)$ shapes with a deterministic algorithm by Kedem et al. [8] 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. The total union for all k sections can thus be computed in $O(kn \log^2(n))$ worst-case time, or $O(kn \log n)$ expected time.

3.3.5 Computing the vertical decomposition of the free space

The free space, which is the complement of the forbidden space just computed, can be divided up into $O(kn)$ cells by extending vertical lines (i.e. V-curves) up and down from the $O(kn)$ curve endpoints until they hit the boundary of the forbidden space. Given the already computed forbidden space, we can easily construct this vertical decomposition with a sweep-line algorithm in $O(nlogn)$ time and $O(n)$ space per section, yielding the decomposition of the complete free space through which we want to find a push plan in $O(kn \log n)$ time.

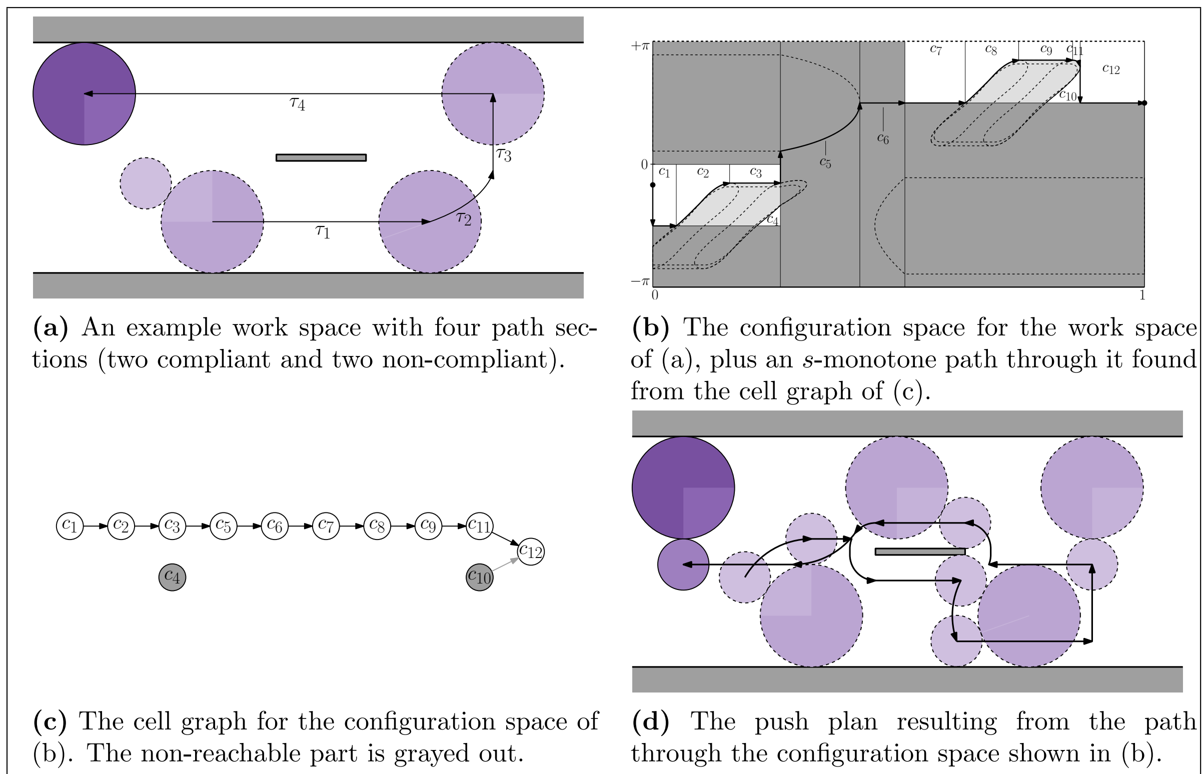
3.4 Finding a contact-preserving push plan

Having computed the configuration space, all that remains is to find a path through the free space. Since moving from right to left in the configuration space means backing up, in effect *pulling* the object rather than pushing, we want to find a path through the free space that never does this. In other words, the path has to be s-monotone. To this end, we direct the connections between cells and their neighbors as pointing from left to right, resulting in a directed acyclic *cell graph* of $O(kn)$ vertices (cells) and edges (connecting neighboring cells).

From this graph we then remove all cells not reachable from the cell containing the initial configuration in $O(kn)$ time, for example by depth-first search. The remaining cells form the *reachable free space*. The rightmost right edge(s) among these cells form the destination configurations, and to find a contact-preserving push plan we simply need to find an s-monotone path through the reachable free space from the initial configuration to any of them. (Figure 3.4(a)–(c) shows an example work space, its configuration space's vertical decomposition into cells, and the resulting cell graph.)

We can solve this problem in two steps: first find a "high-level" path $c_1, c_2, \ldots, c_m$ through the cell graph, and then find "low-level" paths σ_i from the left edge to the right edge of each cell c_i on the high-level path. If we make sure that all σ_i are s-monotone, and that their endpoints are connected, we get a valid contact-preserving push plan. One simple way to do this is to follow the cell boundaries, as Lemmas 3.4 and 3.6 imply that these are all s-monotone. (See Figure 3.4(b)–(d).)

This procedure yields a valid push plan of $O(kn)$ complexity in $O(kn)$ time and space. Even when it's impossible for the object to be pushed to the end of τ due to obstacles, we'll still find a push plan taking it as far along τ as possible. The push plan we find will in general not be very good at minimizing the distance traveled by the pusher, though. Therefore we explore a different method in the next section.



(a) An example work space with four path sections (two compliant and two non-compliant).

(b) The configuration space for the work space of (a), plus an s-monotone path through it found from the cell graph of (c).

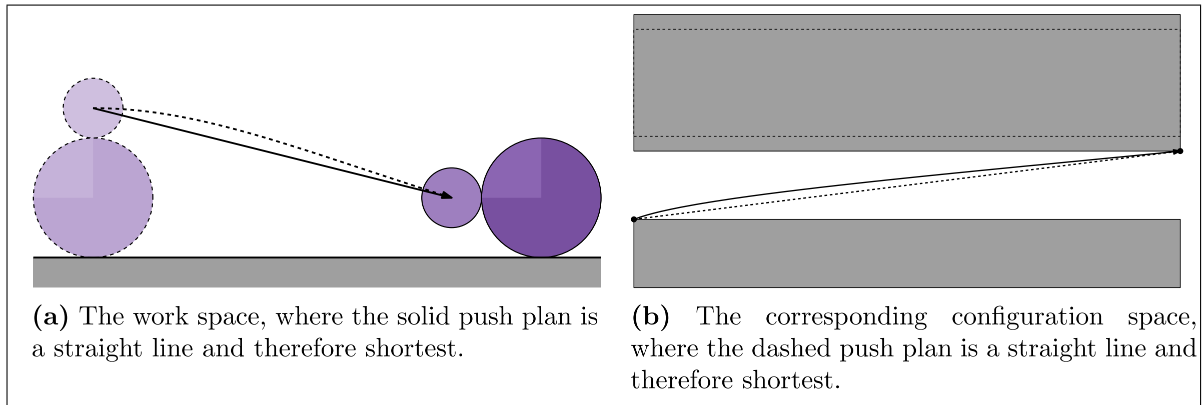
(c) The cell graph for the configuration space of (b). The non-reachable part is grayed out.

(d) The push plan resulting from the path through the configuration space shown in (b).

Figure 3.4: An example showing how our subroutine finds a (not necessarily shortest) contact-preserving push plan.

3.5 Finding a shortest contact-preserving push plan

To find a shortest contact-preserving push plan (in the sense of minimizing the distance traveled by the pusher), we might try to take the reachable free space as computed in the previous section and then apply a standard Euclidean shortest path algorithm. A shortest path is automatically s-monotone because the cell boundaries are s-monotone which means that every detour to the left can be cut short. Unfortunately, a Euclidean shortest path in the configuration space does not necessarily yield a Euclidean shortest push plan in the work space, as shown in Figure 3.5(a)–(b).



(a) The work space, where the solid push plan is a straight line and therefore shortest.

(b) The corresponding configuration space, where the dashed push plan is a straight line and therefore shortest.

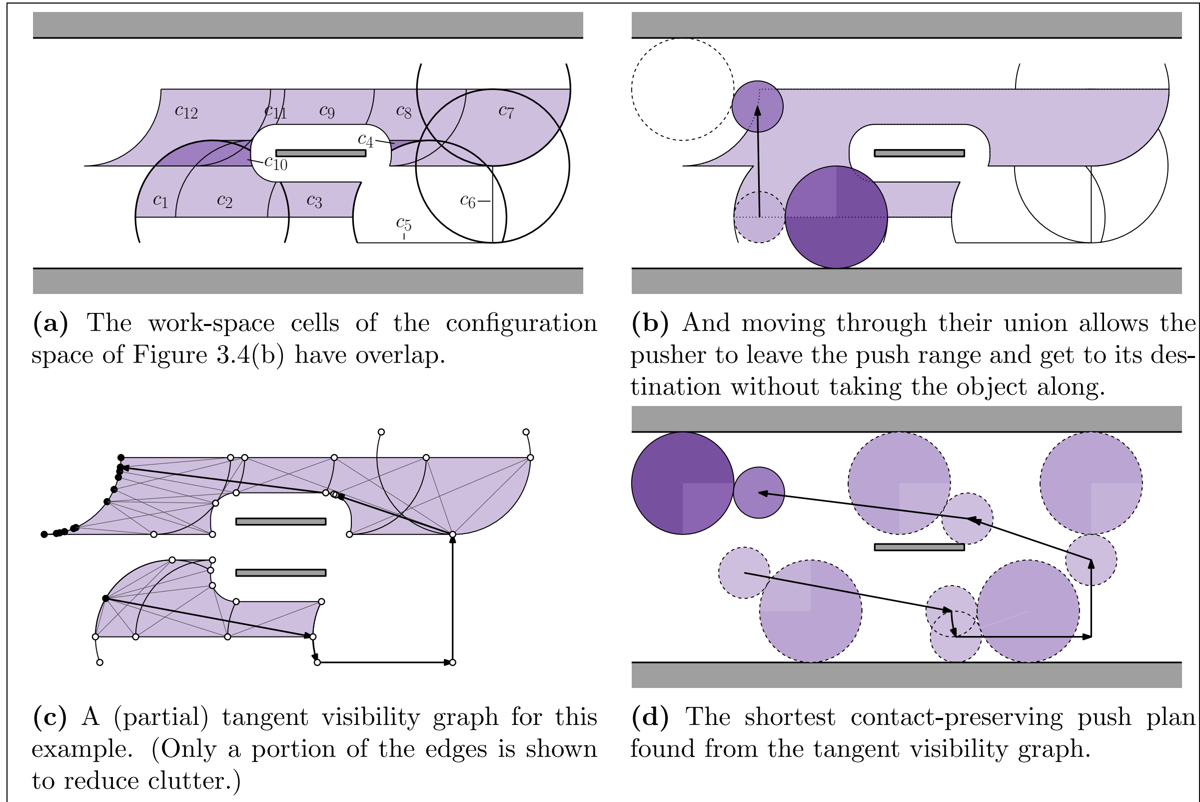
Figure 3.5: Two different push plans (solid and dashed) between a pair of configurations illustrating that a shortest path in the configuration space doesn't necessarily yield the shortest path for the pusher in the work space.

Once again, however, we can perform computations in the work space to solve this configuration-space problem. The curves C bounding a cell c_i of the reachable free space have corresponding simple curves C_p of pusher positions inducing a cell c_i' in the work space. Because of property (A2) of well-behaved path sections, such a work-space cell c_i' is exactly the area through which the pusher can move without intersecting any obstacles and without leaving the push range for the part of τ spanned by c_i. The circular arcs corresponding to the possible contact transits at section boundaries then connect these cells into the space through which the pusher can move to push the object along τ. Figure 3.6(a) shows this space for the example of Figure 3.4.

We have to take care when these work-space cells overlap, though. As illustrated in Figure 3.6(b), making the pusher move through the area formed by the union of all cells and contact transit arcs does not guarantee that the pusher stays in the push range at all times. Instead we have to consider each cell as being in a separate layer so that overlapping cells go over instead of through each other. Letting go of the object will then never allow the pusher to make a shortcut, so a Euclidean shortest path through this $O(kn)$-complexity space from the initial configuration to a destination configurations yields a shortest contact-preserving push plan.

Note that the destination configurations all share the same object position $\mathbf{x}_o$, and the corresponding pusher positions thus form arcs on the circle $\partial(\mathbf{x}_o \oplus D(r_o + r_p))$. Let D be the set of points that are the intersections of this circle with the half-lines from $\mathbf{x}_o$ that either go through an endpoint of, or are tangent to, one of the $O(kn)$ simple curves bounding the

work-space cells. A shortest contact-preserving push plan must then end either in an endpoint of one of the destination arcs, or in one of the points in D. Thus any single-source Euclidean shortest path algorithm capable of dealing with this space bounded by simple curves can be used to find a shortest contact-preserving push plan.



(a) The work-space cells of the configuration space of Figure 3.4(b) have overlap.

(b) And moving through their union allows the pusher to leave the push range and get to its destination without taking the object along.

(c) A (partial) tangent visibility graph for this example. (Only a portion of the edges is shown to reduce clutter.)

(d) The shortest contact-preserving push plan found from the tangent visibility graph.

Figure 3.6: An example showing how our subroutine finds a shortest contact-preserving push plan.

Pocchiola and Vegter [20] describe an algorithm to do this in the presence of convex obstacles, based on *tangent visibility graphs*. A free bitangent of two obstacles is a line segment tangent to both obstacles and not intersecting any (other) obstacles. The initial position, destination positions, and the endpoints of the free bitangents together form the vertices of the tangent visibility graph. Two vertices are connected by an edge if there is a free line segment between them, or they lie on the same obstacle boundary curve. (See Figure 3.6(c).) For m convex obstacles and initial and destination positions there are $q = O(m^2)$ free bitangents, resulting in a graph of $O(m + q)$ vertices and edges that can be computed $O(m \log m + q)$ time. We can then compute a shortest path through the graph by using Dijkstra's algorithm in $O((m + q) \log(m + q))$ time, which yields a shortest push plan through the original space. (See Figure 3.6(d).)

To be able to apply this algorithm, there need to exist m convex obstacles partitioning the plane into our $O(kn)$ work-space cells. Each of our work-space cells is bounded by four convex simple curves however, and if such a curve is oriented the wrong way it would require a concave obstacle. We never need to follow such a curve in a shortest path, however, as it is shorter to go in a straight line between its endpoints. This then does induce a convex obstacle. Hence

we get $m = O(kn)$ convex obstacles from our $O(kn)$ work-space cell boundary curves, and we can find a shortest contact-preserving push plan in $O(k^2n^2 \log(kn))$ time and $O(k^2n^2)$ space.

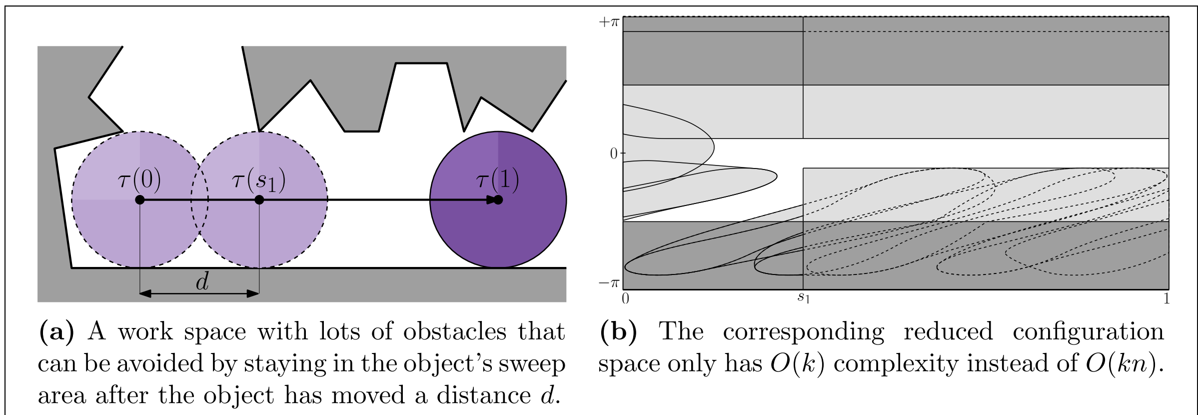
3.6 Low obstacle density

The asymptotic upper bounds derived for our subroutine in the last few sections assume nothing about how densely packed the obstacles are. Realistically, the environment has a low obstacle density, in which case we can sharpen these bounds considerably.

Recall from Subsection 2.4.8 that an environment is a λ-low-density environment when for any disk D of diameter x the number of obstacles $\gamma \in \Gamma$ with length$(\gamma) \geq x$ that intersect D is at most λ. Now let λ be the density of the given environment, i.e. the smallest λ for which it is a λ-low-density environment, and let σ be the length of the smallest obstacle. A disk with diameter σ then intersects at most λ obstacles.

By definition property (A4) 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 initial stretch of length d always fits inside a disk with diameter $d + 2r_o + 4r_p = O(r_o)$, and thus can intersect at most $O(\lambda \cdot {r_o}^2/\sigma^2)$ obstacles (because it can be covered by $O({r_o}^2/\sigma^2)$ disks of diameter σ).

For constant λ and $\sigma = \Omega(r_o)$ this means the configuration space for any length-d prefix of a path section has complexity $O(1)$. The complexity of the remaining suffix of the path section can still be $\Theta(n)$ however (as was also hinted at in Subsection 2.4.8), thus if we insist on computing a shortest push plan we can't lower the complexity of the configuration space in the worst case. If we drop that requirement, then for this suffix of the path section 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 shape can be computed in $O(1)$ time by property (A4).) The total complexity of this *reduced configuration space* is only $O(k)$. (See Figure 3.7.)



(a) A work space with lots of obstacles that can be avoided by staying in the object's sweep area after the object has moved a distance d.

(b) The corresponding reduced configuration space only has $O(k)$ complexity instead of $O(kn)$.

Figure 3.7: An example of a work space and its reduced configuration space.

For each path section we then only have $O(1)$ shapes to compute the union and vertical decomposition of, bringing the total time needed for that down to $O(k)$. Computing these

shapes section-by-section, obstacle-by-obstacle as described in Subsection 3.3.2 still takes $O(kn)$ time, though. We can do this more efficiently by computing them for all path sections and obstacles at once, using Balaban's algorithm [3] for the intersection computations. This brings the time to compute the configuration space down to $O((k+n)\log(k+n))$, and the space to $O(k+n)$.

As the decomposition of the free space now only has $O(k)$ cells, the path-finding method of Section 3.4 takes $O(k)$ time and space. The method of Section 3.5 would take $O(k^2 \log k)$ time and $O(k^2)$ space, but wouldn't necessarily yield an optimal solution anymore. It does, however, still yield a "quasi-optimal" solution that is shortest among all contact-preserving push plans that always keep the pusher in the object's sweep area (except possibly in the length-d prefixes of sections). Table 3.1 summarizes these results.

	High obstacle density	Low obstacle density
Computing the configuration space		
- constructing all $\mathcal{C}_{\gamma,i}$	kn	$(k+n)\log(k+n)$
- constructing all FPR_i	k	k
- taking their union	$kn \log n$ $^{(*)}$	k
- vertical decomposition	$kn \log n$	k
Finding any path	kn	k
Finding a shortest path	$k^2 n^2 \log(kn)$	$k^2 \log k$ $^{(**)}$

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

$^{(**)}$ This yields a "quasi-optimal" solution.

Table 3.1: Asymptotic upper bounds for the running time of our subroutine for constructing a contact-preserving push plan amongst n obstacles given an object path of k well-behaved path sections.

Chapter 4

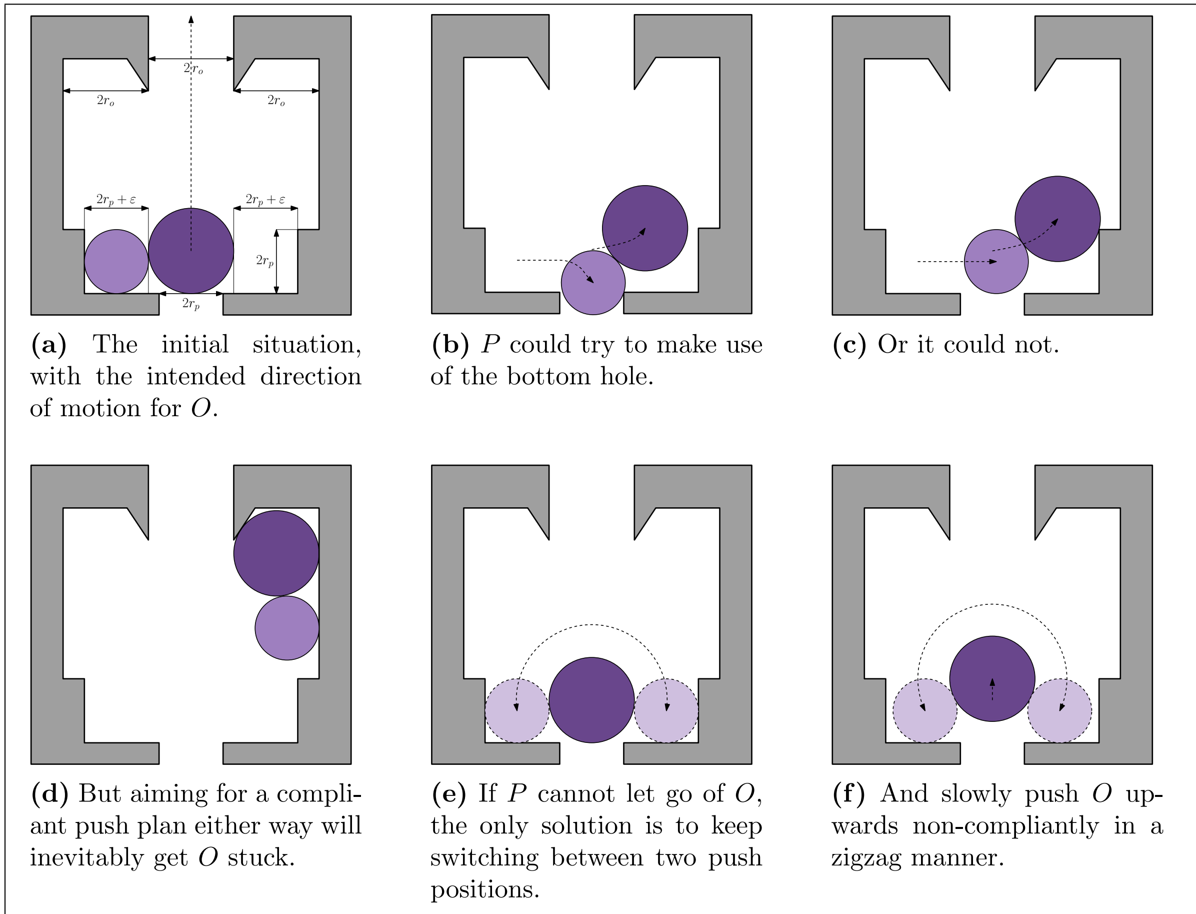
Pushing and releasing

Up until now we've only considered contact-preserving push plans, i.e. push plans in which the pusher always maintains contact with the object. It turns out that this can lead to unnecessarily long or complex push plans. Furthermore, it may even prevent a push plan to be found altogether, even when there exists a perfectly good unrestricted push plan where the pusher does occasionally release the object. In this chapter we first substantiate these claims by considering examples (using Nieuwenhuisen's path sections) where these phenomena occur. We then adapt our subroutine for contact-preserving push plans to find unrestricted push plans in such cases (for well-behaved path sections in general).

4.1 Releasing can be beneficial

Nieuwenhuisen describes an example where a simple push plan using compliance is impossible, and only a complex one without compliance can be used [15, Section 7.7]. Figure 4.1 is a slight variation of that example, where there *is* a simple push plan, but no simple *contact-preserving* push plan.

Figure 4.1(a) shows the initial situation and the intended direction of motion for the object. (We don't require that this path is followed exactly, as long as the object gets pushed out of the room.) The problem here is that P cannot get "below" O without pushing it against a wall, where it will inevitably get stuck as shown in Figure 4.1(b)–(d). When contact between P and O has to be maintained, the only possibility is to keep switching between two pushing positions on either side of O and slowly push O up a little bit at a time, as shown in Figure 4.1(e)–(f). When the pusher *is* allowed to let go of the object, a much simpler push plan is possible, as shown in Figure 4.2.



(a) The initial situation, with the intended direction of motion for O.

(b) P could try to make use of the bottom hole.

(c) Or it could not.

(d) But aiming for a compliant push plan either way will inevitably get O stuck.

(e) If P cannot let go of O, the only solution is to keep switching between two push positions.

(f) And slowly push O upwards non-compliantly in a zigzag manner.

Figure 4.1: An example in which keeping contact between the pusher and the object makes a very complicated push plan necessary.

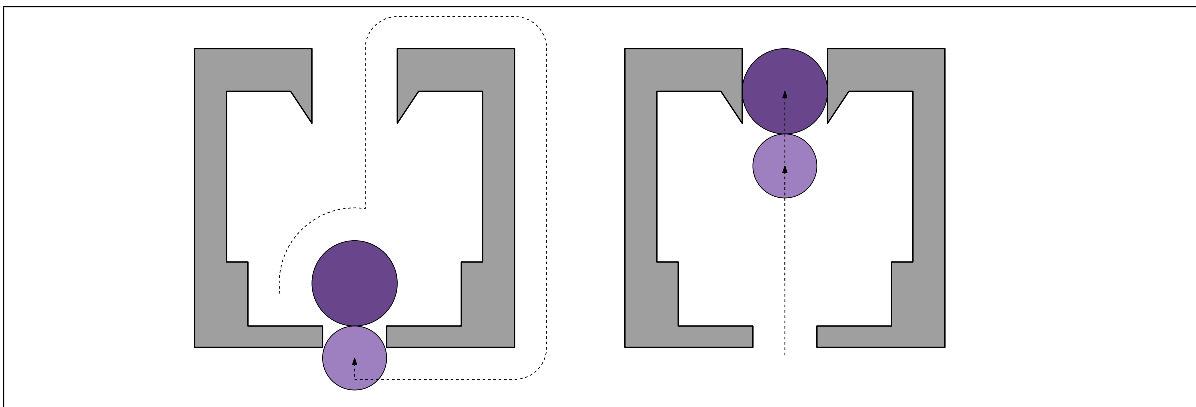
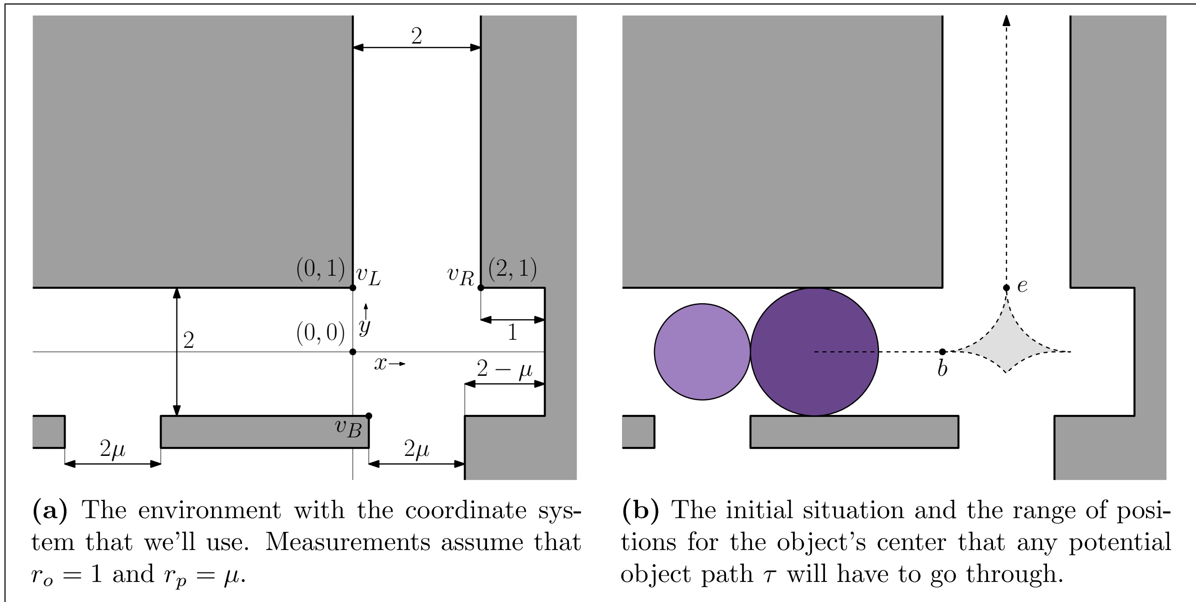


Figure 4.2: When the pusher is allowed to release the object, a much simpler push plan is possible.

4.2 Releasing can be necessary

There are even cases where no push plan exists at all, whether complex or not, unless the pusher is allowed to let go of the object. Consider the environment in Figure 4.3(a).

We 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 the figure, around the bend, to the upper right part of the figure. For this there is a range of potential paths τ which O could be made to follow, as seen in Figure 4.3(b).



(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 range of positions for the object's center that any potential object path τ will have to go through.

Figure 4.3: An example where it's impossible to get the object to its destination position with a contact-preserving push plan.

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, i.e. the subpath from point b to point e in Figure 4.3(b).

In particular, we'll look at the *highest* possible such subpath, i.e. the subpath that maximizes the distance traveled in the positive y direction for the distance traveled in the positive x direction. In principle, this would be the circular compliant motion around vertex v_L, however that motion is not always possible, as stated in the following lemma.

Lemma 4.1. *In the example of Figure 4.3, the pusher can push the object compliantly around v_L from b to e if and only if $\mu \leq 1/3$.*

Proof. In this motion, the object's position describes a circular arc around v_L with radius equal to the object's. The area swept out by this is obstacle free. However, the combination of pusher and object sweep out a larger area. As discussed in Subsection 2.4.3, this area is minimal exactly when the pusher's trajectory is a circular arc around v_L with radius

$\sqrt{r_o{}^2 + (r_o + r_p)^2} = \sqrt{\mu^2 + 2\mu + 2}$. The pusher can accomplish the object's compliant motion if and only if this minimal sweep area is obstacle free, because all other potential sweep areas for this motion are supersets.

In the example, that area is obstacle free exactly when $\sqrt{\mu^2 + 2\mu + 2} \leq 2 - \mu$, because P can only go as low as $2 - \mu$ below the vertex before hitting the bottom wall. Squaring both sides of this inequality and simplifying yields $\mu \leq 1/3$. □

From now on, assume that $\mu > 1/3$. Lemma 4.1 then implies that if the pusher tries to push the object compliantly around v_L, the pusher will hit the bottom wall at some point, and the object will be at some point m_1 between b and e. From b to m_1, the object will have turned a certain angle φ ($0 < \varphi < \pi/2$) around v_L, where (see Figure 4.4):

$$\begin{aligned} &\varphi = a - b \\ &\sin(a) = \frac{2-\mu}{\sqrt{\mu^2+2\mu+2}} \qquad \cos(a) = \frac{\sqrt{6\mu-2}}{\sqrt{\mu^2+2\mu+2}} \\ &\sin(b) = \frac{1}{\sqrt{\mu^2+2\mu+2}} \qquad \cos(b) = \frac{1+\mu}{\sqrt{\mu^2+2\mu+2}} \end{aligned} \tag{4.1}$$

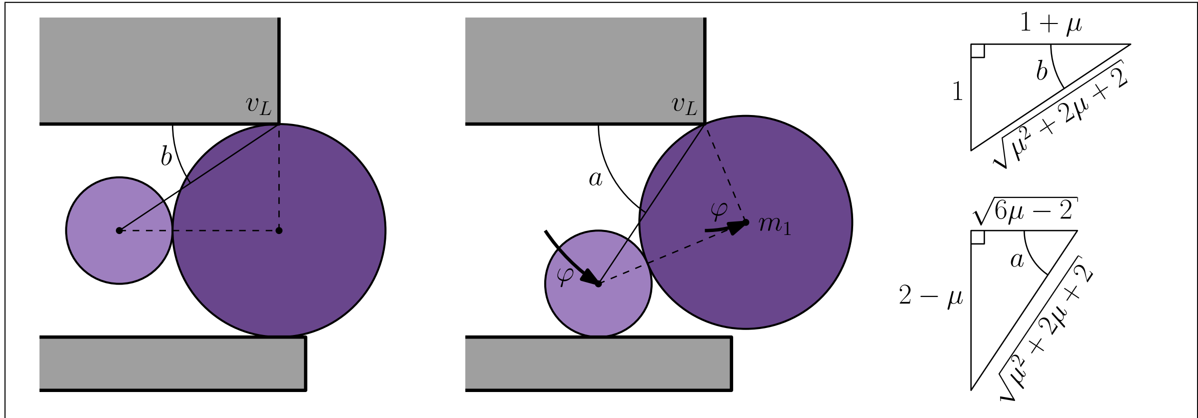


Figure 4.4: If P's radius is more than one third that of O, it will get stuck trying to push O around the inner corner vertex v_L after an angle $\varphi = a - b$.

From m_1 onward, the object will have to be pushed away from v_L, and move non-compliantly. The highest possible path that does this, is the path that makes the pusher move compliantly along the bottom wall, thus maximizing the pushing angle. As discussed in Section 2.2, this makes the object follow a hockey-stick curve. As usual we denote the push angle, i.e. the angle that the line from P's to O's center makes with the positive x-axis, by θ. Then the coordinates of O when following this curve can be expressed as a function of θ, for $\varphi \leq \theta \leq \pi/2$:

$$\begin{aligned} x(\theta) &= (1+\mu)\ln\left(\frac{\tan(\theta/2)}{\tan(\varphi/2)}\right) + (1+\mu)\cos(\theta) - (1+\mu)\cos(\varphi) + \sin(\varphi) \\ y(\theta) &= (1+\mu)\sin(\theta) - 1 + \mu \end{aligned} \tag{4.2}$$

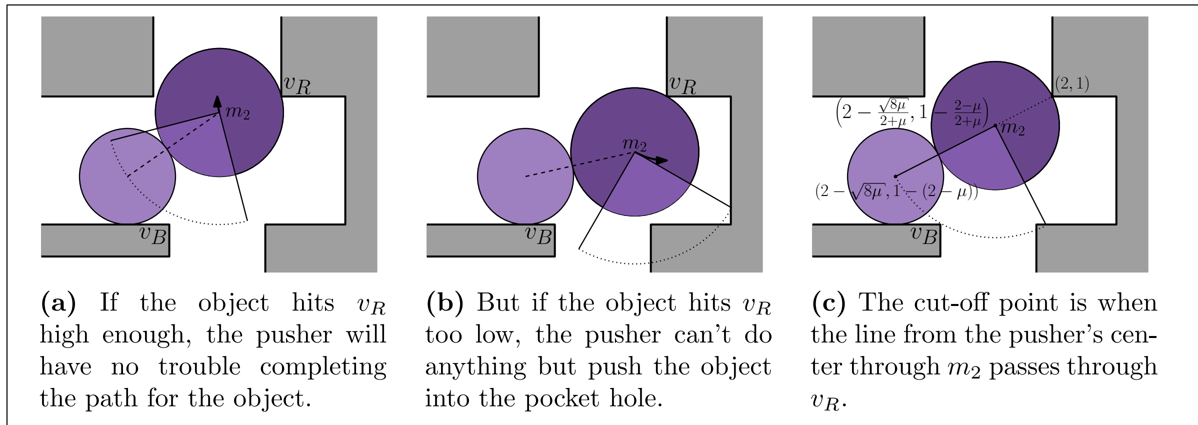
Lemma 4.2. *Following the hockey stick curve from m_1 on, the object will eventually hit the outer corner vertex v_R, at some point m_2.*

When m_2 is above the line $y = 1 - \frac{2-\mu}{2+\mu}$, the pusher can push the object to e by a circular compliant motion around v_R.

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, 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. Therefore the object inevitably bumps into v_R at some point m_2 along the 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 wall, as well as the push range which would push the object upwards compliantly around v_R. If m_2 is high enough, θ will lie in this range, as seen in Figure 4.5(a).



(a) If the object hits v_R high enough, the pusher will have 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.

Figure 4.5: Being pushed along the hockey stick curve, the object will eventually hit the outer corner vertex v_R. Where this collision occurs determines whether the object will get stuck.

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 leads the object into the pocket hole, as seen in Figure 4.5(b).

The cut-off point where θ lies just barely 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 Figure 4.5(c). □

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

Theorem 4.3. *There is a μ_c, $1/3 < \mu_c < 1$, such that for all $\mu > \mu_c$, the example of Figure 4.3 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).*

Proof. From Equations (4.2) it follows that the hockey stick curve intersects the line $y = 1 - \frac{2-\mu}{2+\mu}$ exactly when:

$$\sin(\theta) = \frac{2-\mu}{2+\mu} \tag{4.3}$$

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:

$$x(\theta) \geq 2 - \frac{\sqrt{8\mu}}{2+\mu} \tag{4.4}$$

Combining Equations (4.1) through (4.4) this condition becomes:

$$\ln(f(\mu)) \geq g(\mu), \tag{4.5}$$

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

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

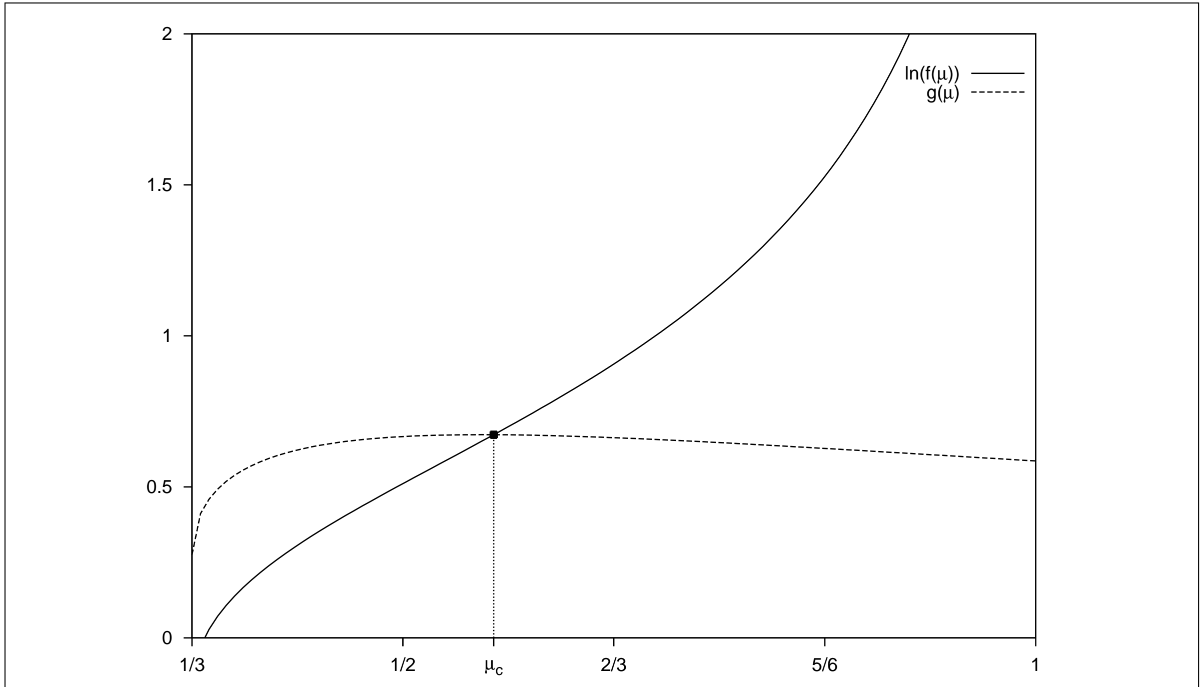


Figure 4.6: The smallest μ value for which the example of Figure 4.3 admits no contact-preserving push plan.

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 Figure 4.6. 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 all μ between 0 and 1, even those greater than μ_c, there is always a push plan when the pusher is allowed to release the object, as shown in Figure 4.7. □

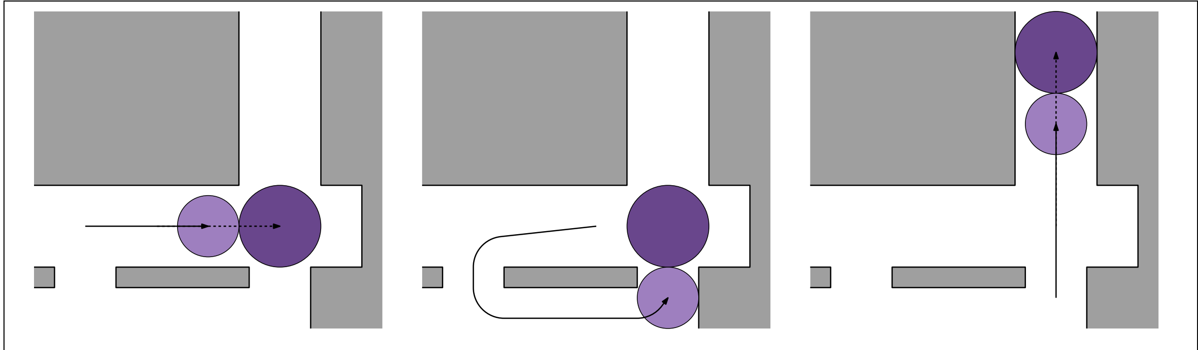


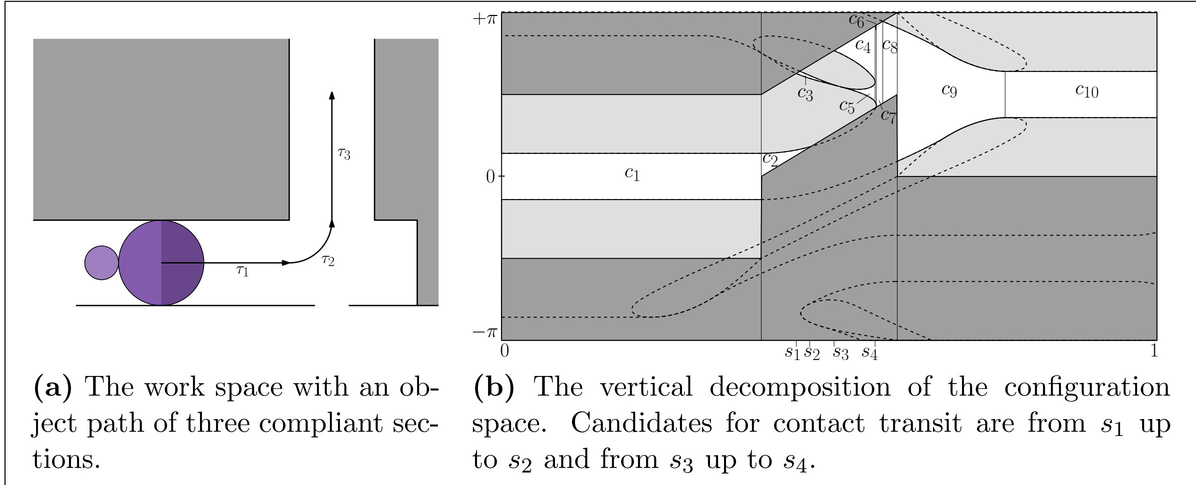
Figure 4.7: When the pusher is allowed to release the object, a push plan *does* exist.

Thus in order to always find a push plan where one exists we'll have to adapt our subroutine to allow the pusher to let go of the object occasionally.

4.3 Canonical releasing positions

Whenever the valid positions for the pusher consist of one contiguous range it makes little sense to release the object. To reach a different pushing angle a simple contact transit will do just as well, and this will always yield a shorter path for the pusher. Thus the object positions at which the pusher may want to release the object are those for which there are two or more unconnected ranges of pushing positions. In the configuration space this corresponds with a vertical line intersecting two or more cells of the free space. In the example of Figure 4.8, such positions are $\tau(s_1)$ up to $\tau(s_2)$ and $\tau(s_3)$ up to $\tau(s_4)$.

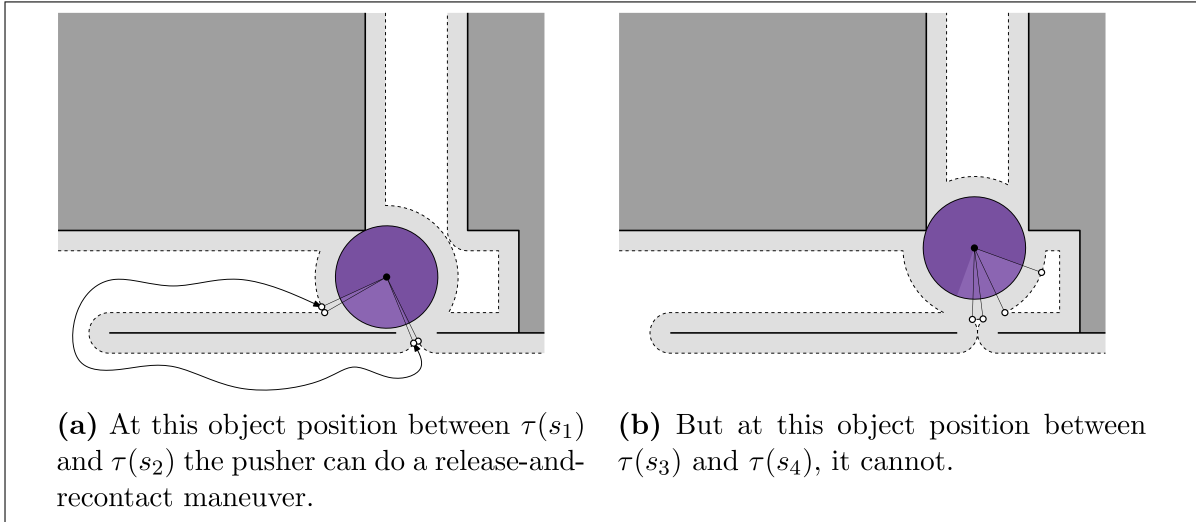
At such a candidate position $\tau(r)$ a release-and-recontact is possible only when a path exists between these two pusher locations that does not intersect $U(r_p) \cup (\tau(r) \oplus D(r_o + r_p))$. Figure 4.9(a) shows a candidate position from the above example where a release-and-recontact is possible, and Figure 4.9(b) shows a candidate position from the same example where it is not possible. In general it's impossible to check all candidate positions for feasibility in this way as there are already infinitely many candidates in this simple example. Fortunately, we can reduce our candidate set to a finite number of positions, as follows.



(a) The work space with an object path of three compliant sections.

(b) The vertical decomposition of the configuration space. Candidates for contact transit are from s_1 up to s_2 and from s_3 up to s_4.

Figure 4.8: An example where releasing the object is necessary to find a push plan, and the candidates for positions at which to do so.



(a) At this object position between $\tau(s_1)$ and $\tau(s_2)$ the pusher can do a release-and-recontact maneuver.

(b) But at this object position between $\tau(s_3)$ and $\tau(s_4)$, it cannot.

Figure 4.9: Finding out whether a release-and-recontact maneuver can be performed for a certain candidate position is a traditional translational path-finding problem.

We define the *canonical releasing positions* to be the set of object positions for which the vertical slice in the configuration space not only intersects the free space in at least two intervals, but also goes through the leftmost or rightmost point of a cell or of a component of some $\mathcal{C}_{\gamma,i}$. That this set of $O(kn)$ positions suffices is expressed by the following theorem:

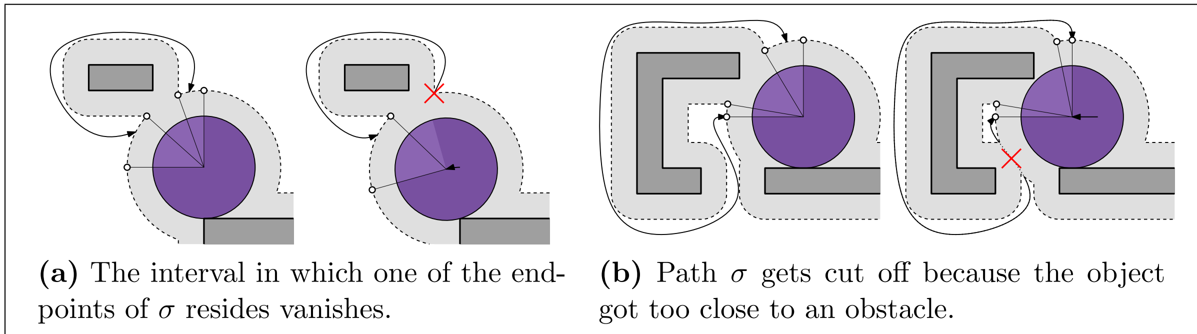
Theorem 4.4. *If an unrestricted push plan exists for a given input, then there is also an unrestricted push plan where all releases happen at canonical releasing positions as described above.*

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 the 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' with $0 \leq r' < r$ and no other canonical releasing positions lying 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 the path resides shrinks to a point and vanishes (see Figure 4.10(a)), or the path gets cut off between the obstacles and the object (see Figure 4.10(b)). For the former, we have to pass the leftmost point of c_1 or c_2, as such an interval corresponds to a vertical slice of these cells. For the latter, the distance between the object and some obstacle γ has to shrink from being greater than $r_o + 2r_p$ to being smaller. As mentioned in the proof of Lemma 3.4, the intervals of the path section where this distance is smaller than $r_o + 2r_p$ correspond to the components of $\mathcal{C}_{\gamma,i}$. Thus for this situation to occur we have to pass the rightmost point of some component of $\mathcal{C}_{\gamma,i}$. That means that in both cases we have to pass a canonical releasing point between r and r', in contradiction with our assumption. □



(a) The interval in which one of the endpoints of σ resides vanishes.

(b) Path σ gets cut off because the object got too close to an obstacle.

Figure 4.10: The two situations in which moving backwards along τ would make us unable to maintain a release-and-recontact path σ.

This theorem leads fairly straightforwardly to a method for finding unrestricted push plans by testing such canonical releasing positions for feasibility, as will be described in the next section. Note, however, that this method may not always yield a *shortest* unrestricted push plan. That a release and recontact can be done anywhere in an interval between two canonical releasing positions doesn't mean that all of these choices lead to an equally good push plan, and the optimal choice might not be on either of the endpoints.

4.4 Finding an unrestricted push plan

To find an unrestricted push plan we proceed in mostly the same way as for the contact-preserving setting, as described in Section 3.4. Once we have computed the cell graph of the free space, however, we first extend it with additional edges for release-and-recontact maneuvers before finding a path through it.

4.4.1 Computing the canonical releasing points

In constructing the vertical decomposition of the free space we already computed the leftmost and rightmost points of the cells and configuration-obstacle components. Let r be the s-coordinate of one of these $O(kn)$ points. We can find the set $C_r = \{c_1, \ldots, c_m\}$ of cells intersected by the vertical line $s = r$ in $O(\log(kn) + m)$ time using the search structure for point location associated with the vertical decomposition of the free space. Whenever $m > 1$ we know that r is a canonical releasing point.

4.4.2 Extending the cell graph

To see between which of the m cells release-and-recontact maneuvers can actually be performed, we compute $UO(r) = U(r_p) \cup (\tau(r) \oplus D(r_o + r_p))$ and the vertical decomposition of its complement $\overline{UO(r)}$. We do this by first computing $U(r_p) = \bigcup_{\gamma\in\Gamma}(\gamma \oplus D(r_p))$ (with a deterministic algorithm by Kedem et al. [8] in $O(n \log^2 n)$ time, or a randomized incremental algorithm by Miller and Sharir [14] in $O(n \log n)$ expected time) and the vertical decomposition of $\overline{U(r_p)}$ (with a simple sweep-line algorithm). After this one-time preprocessing of the obstacles we can compute the vertical decomposition of $\overline{UO(r)}$ for any r in $O(n)$ time.

For each cell $c_i \in C_r$ we then take a point p_i on the intersection of c_i with the line $s = r$ and locate to which component of $\overline{UO(r)}$ the associated pusher position belongs. This can be done in $O(\log n)$ time using the point location search structure of $\overline{UO(r)}$'s vertical decomposition. Between any two pusher positions found to be in the same component there is a possible release-and-recontact maneuver. Thus the (at most $m - 1$) extra edges of the cell graph for a canonical releasing point r can be computed in $O(n + m \log n)$ time.

Because all $\mathcal{C}_{\gamma,i}$ and FPR_i are s-monotone (Lemmas 3.4 and 3.6) we know that $m \leq n$ always holds. Thus computing the $O(kn)$ canonical releasing points and extending the cell graph will produce at most $O(kn^2)$ extra edges in $O(kn \log(kn) + kn^2 \log n)$ total time.

4.4.3 Computing the high-level path

In the cell graph extended with these extra edges we then find a high-level path using depth-first search as before. The edges traversed in this path can now correspond both to contact transits and to release-and-recontact maneuvers. Whenever two consecutive edges (c_1, c_2) and (c_2, c_3) of the path are both release-and-recontact maneuvers for the same releasing point we collapse them into a single edge (c_1, c_3). Hereby we prevent doing $\Omega(kn^2)$ release-and-recontact maneuvers where $O(kn)$ would have sufficed.

4.4.4 Computing the connecting low-level paths

Finally, we compute the low-level paths through the cells on this high-level path to arrive at the final push plan. Again we follow the cell boundaries whenever the pusher maintains contact with the object. An $O(n)$-complexity low-level path for a release-and-recontact maneuver at

releasing point r can be computed in $O(n)$ time by searching the roadmap of $\overline{UO(r)}$'s vertical decomposition.

Thus we can compute a $O(kn^2)$-complexity unrestricted push path from the extended cell graph in $O(kn^2)$ time and space.

4.5 Low obstacle density

Under the low-obstacle-density assumptions of Section 3.6, using the reduced configuration space defined there, there are only $O(k)$ canonical releasing points which can be computed in $O(k \log k)$ time. Each of these intersects only $m = O(1)$ cells of the free space, so we extend the cell graph with only $O(k)$ extra edges in $O(kn)$ time. A high-level path through this graph can be found in $O(k)$ time, for which we can find the connecting low-level paths in $O(kn)$ time. This yields an $O(kn)$-complexity unrestricted push plan in $O((k+n)\log(k+n)+kn)$ time and $O(kn)$ space in total.

Table 4.1 summarizes these results.

	High obstacle density	Low obstacle density
Preprocessing the obstacles	$n \log n$ (*)	$n \log n$ (*)
Computing the configuration space	$kn \log n$ (*)	$(k+n)\log(k+n)$
Finding any path		
- computing the canonical releasing points	$kn\log(kn)+kn^2$	$k \log k$
- extending the cell graph	$kn^2 \log n$	kn
- computing the high-level path	kn^2	k
- computing the connecting low-level paths	kn^2	kn

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

Table 4.1: Asymptotic upper bounds for the running time of our subroutine for constructing an unrestricted push plan amongst n obstacles given an object path of k well-behaved path sections.

Chapter 5

Conclusion

In this thesis we have studied the manipulation path-planning problem of a disk (the pusher) pushing another disk (the object) amongst non-intersecting line-segment obstacles in the plane. We have introduced the notions of contact-preserving push plans and unrestricted push plans, the former being solutions to the problem where the pusher maintains contact with the object at all times, the latter ones where the pusher may occasionally let go of the object. In addition, we have shown that there are cases where simple unrestricted push plans exist even though no contact-preserving push plans exist.

We have briefly discussed the prior work by Nieuwenhuisen [15] that reduces the problem of finding push plans to that of planning a path for the pusher only, with the path of the object already given. Nieuwenhuisen's subroutine for solving this subproblem was discussed in-depth, and we point out that it produces only contact-preserving push plans, as well as having a few other shortcomings. Both issues were resolved in the new subroutine developed in this thesis. We first summarize these improvements, and then list some directions for further research.

5.1 Improvements over prior work

- Our method abstracts away from the four types of path sections handled by Nieuwenhuisen's subroutine and captures them in a single, more general notion of well-behaved path sections. We accomplished this while still matching the running time of Nieuwenhuisen's subroutine, and using less space and preprocessing time (see Table 5.1).

  It may seem that this generalization is a simplifying convenience only, as Nieuwenhuisen's path sections already capture the physics of applying this method to real-world robotics. However, consider the problem of *pulling* the object to a destination instead of pushing it. Path sections for this problem are also well-behaved (except it's a suffix instead of a prefix of any path section where the puller can stay in the object's sweep area), and our method can thus be used to solve the pulling problem as well.

- Nieuwenhuisen's subroutine finds only contact-preserving push plans, whereas ours can also find unrestricted push plans. In addition, Nieuwenhuisen's push plans don't optimize any particular criterion, whereas we can find shortest contact-preserving push

plans that minimize the distance traveled by the pusher. These features do come with an additional cost, though (see Table 5.1).

- We've assumed throughout that the pusher is smaller than the object, i.e. $r_p < r_o$, but only used this fact as a means to provide lower time bounds for environments with low obstacle density. Our subroutine for the high-obstacle-density case can be used with $r_p \geq r_o$ without modification. Nieuwenhuisen's subroutine could also be made to handle this case, but not without work.

- We've assumed that our obstacles are line segments, but some forms of curved obstacles can be handled as well. All that's really needed is that $\gamma \oplus D(r)$ is an $O(1)$-complexity shape bounded by simple curves for every obstacle γ. To find shortest contact-preserving push plans we additionally need that these shapes are convex, and to guarantee the same time bounds as before we need $\{\gamma \oplus D(r) \mid \gamma \in \Gamma\}$ to be a collection of pseudodisks so that $U(r)$ again has $O(n)$ complexity.

	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	—	$k^2 n^2 \log(kn)$	—	$(k+n)\log(k+n) + k^2 \log k$ (**)
Any UPP	—	$kn \log(kn) + kn^2 \log n$	—	$(k+n)\log(k+n) + kn$

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

(**) This yields a "quasi-optimal" solution as discussed in Section 3.6.

Table 5.1: A comparison of the asymptotic upper bounds for the running time between Nieuwenhuisen's subroutine and ours for constructing contact-preserving push plans (CPPP) and unrestricted push plans (UPP) for Nieuwenhuisen's path sections.

5.2 Further research

- Although non-line-segment obstacles can be handled fairly easily, having the pusher and/or object be some shape other than a disk is not as simple. A pushing motion may rotate the object and/or pusher, so we'll have to keep track of their orientations as well as their positions. 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 will still not suffice.

- We have not attempted to solve the problem of finding shortest unrestricted push plans. It may be that a different finite set of canonical releasing points can be defined, for which it can be proven that a shortest unrestricted push plan always exists that only does releases at these points. If this is the case, it may be possible to adapt our method for finding shortest contact-preserving push plans to find shortest unrestricted push plans.

- We have studied the subproblem where the path followed by the object is already given, mentioning that a subroutine for this can be used in Nieuwenhuisen's algorithm based on Rapidly-exploring Random Trees [10] to solve the global problem where only an initial and destination position are given. This algorithm is only probabilistically complete, however, and it depends on the fact that the pusher is smaller than the object. Nieuwenhuisen [18] mentions that it might be possible to adapt it into an exact, complete algorithm by a kind of visibility graph approach.

  An alternative may be to study the 3-dimensional configuration space induced by allowing one disk to move freely while the other is constrained to maintain contact with it. Defining 3-dimensional configuration-space obstacles for this space is fairly simple, but there is no straightforward analogue to our forbidden push ranges. It may be possible to use some form of constrained path finding instead, but we have not explored this possibility.

- The worst-case running times we derived assume we can encounter pathological cases in our input environments. We discussed how more realistic inputs having low obstacle density lower the complexity of the configuration space. There may be additional (but still realistic) assumptions one can make to lower the complexity of the tangent visibility graph used to find shortest contact-preserving push plans, or the complexity of the paths for release-and-recontact maneuvers. Also, perhaps one can use such additional assumptions to guarantee a lower complexity for our unreduced configuration space, so that we can efficiently find shortest (instead of just "quasi-shortest") contact-preserving push plans.

- If the work-space cells would be bounded by line segments instead of convex simple curves, then they would form a simple polygon and hence would allow the computation of a shortest path tree in linear time [7]. Perhaps such an approach could be adapted to work with the "simple curved polygon" formed by our work-space cells, yielding a much faster method of computing shortest contact-preserving push plans.

Bibliography

[1] P.K. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 4, pages 3124–3129, 1997.

[2] H. Arai and O. Khatib. Experiments with dynamic skills. In *Proceedings of the 1994 Japan-USA Symposium on Flexible Automation*, pages 81–84, 1994.

[3] I.L. Balaban. An optimal algorithm for finding segment intersections. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, pages 211–219, 1995.

[4] M. de Berg, M.J. Katz, A.F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 294–303, 1997.

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

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

[7] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proceedings of the 2nd Annual ACM Symposium on Computational Geometry*, pages 1–13, 1986.

[8] 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*, 1:59–70, 1986.

[9] V. Koltun. Segment intersection searching problems in general settings. In *Proceedings of the 17th Annual ACM Symposium on Computational Geometry*, pages 197–206, 2001.

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

[11] K.M. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

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

[13] M.T. Mason and K. Lynch. Dynamic manipulation. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 152–159, 1993.

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

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

[16] D. Nieuwenhuisen, A.F. van der Stappen, and M.H. Overmars. Path planning for pushing a disk using compliance. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4061–4067, 2005.

[17] D. Nieuwenhuisen, A.F. van der Stappen, and M.H. Overmars. Pushing using compliance. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2010–2016, 2006.

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

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

[20] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 248–257, 1995.