Geometric design of part feeders

Geometric design of part feeders

Geometrisch ontwerpen van onderdeelvoeders

(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van
doctor aan de Universiteit Utrecht
op gezag van Rector Magnificus, Prof. Dr. H. O. Voorma,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen
op maandag 4 december 2000 des middags te 12:45 uur

door

Robert-Paul Mario Berretty
geboren op 19 december 1972, te Nijmegen

promotor: Prof. dr. M. H. Overmars
Faculteit Wiskunde & Informatica
co-promotor: Dr. ir. A. F. van der Stappen
Faculteit Wiskunde & Informatica

ISBN 90 – 393 – 2540 – 5

The research in this thesis was supported by the Netherlands Organization for Scientific Research (N.W.O.).

Preface

This thesis considers problems derived from industrial engineering, and treats them in a rather theoretical setting, called "computational geometry". This work has been carried out within the research group of Mark Overmars, which is centered around applied algorithms, i.e. solving problems that are inspired by practical implications.

My first introduction to computational geometry was very pleasant. I discovered that the combination of geometry and algorithms was a perfect fit for me. I decided to ask Mark Overmars, who taught the course on computational geometry, to be my supervisor for my master's degree. This turned out to be a lucky choice, and I almost automatically got involved as a PhD student in a joint research project with the industrial engineering department from U.C. Berkeley. The aim of our joint research project was to bring together computational geometry and the problems in engineering, in order to try to understand the nature of the problems, and come up with general solutions, as opposed to the ad-hoc, trial-and-error based current practices. I consider myself fortunate for having been given the opportunity to do theoretical research on such appealing problems.

I visited Ken Goldberg and his students at the Industrial Engineering and Operations Research department at U.C. Berkeley twice during my PhD. The visits were both fruitful and enjoyable. I always went back home full of new ideas and problems to think of, and I enjoyed seeing how some of our ideas really worked in practice. In Chapter 6, I include a picture of a real (industrial) device that was built by Ken's students, and evolved from our joint efforts.

At the department in Utrecht, I was happy to find a young and lively group of computational geometers. Within this group, my position as a PhD student was

created to continue the work of Frank van der Stappen. During my PhD, he was my co-supervisor. In Utrecht Mark, Frank and I spent a lot of time together thinking deep thoughts.
The result of our joint efforts and a lot of perspiration finally evolved into this thesis. I hope that the reader can taste the pleasure that I had writing it.

RP
Utrecht, October 2000

Contents

Preface **v**

1 Introduction **1**
1.1 Automated assembly 2
1.2 Thesis outline 7

2 Pushing **11**
2.1 Mechanics of pushing 11
2.1.1 Friction 12
2.1.2 Planar sliding 13
2.2 Geometric modeling 15
2.2.1 The push function 16
2.2.2 Orienting planar parts by pushing 18
2.2.3 From a push function to a part 22

3 Fence design **25**
3.1 Geometric modeling 27
3.2 Computing fence designs 29
3.3 An output-sensitive algorithm 33
3.3.1 Maintaining the shortest intervals 33
3.3.2 Updating in $O(n \log n)$ time 35
3.3.3 Incremental computation of the fence design 38
3.4 Completeness for asymmetric parts 39
3.5 Arbitrary parts 40

3.6 Friction 44
3.6.1 The model 44
3.6.2 Algorithms for fence designs with friction 45
3.7 Modular fence designs 47
3.7.1 The model 47
3.7.2 Algorithms for modular fence designs 49
3.8 Implementation 51
3.9 Discussion 51

4 Inside-out pulling 55
4.1 Geometric modeling 56
4.1.1 The effect of pulling a part 57
4.1.2 The pull function 59
4.1.3 Properties of the pull functions 60
4.2 Completeness of pulling 66
4.3 Computing optimal pull plans 67
4.4 Arbitrary polygonal parts 68
4.5 Discussion 69

5 Pushing into the third dimension 73
5.1 Pushing parts 74
5.1.1 Modeling the feeder 75
5.1.2 The push function 76
5.2 Orienting a polyhedral part 81
5.3 Computing a push plan 84
5.4 Plates with fences 89
5.5 Discussion 90

6 Traps for vibratory bowl feeders 93
6.1 Geometric Modeling 94
6.2 Analyzing a trap 96
6.2.1 Arbitrary polygonal traps and parts 97
6.2.2 Convex traps and parts 100
6.3 Design of traps 103
6.3.1 Balconies 104
6.3.2 Gaps 106
6.3.3 Canyons 110
6.3.4 Slots 114
6.3.5 Arbitrary polygonal traps 116

6.4 Experimental results 120
6.5 Discussion 120

7 Concluding remarks **123**

A Fence designs for arbitrary parts **127**
A.1 Acyclic short environments 127
A.2 Cyclic environments 133
A.2.1 Preliminaries 133
A.2.2 The reduce 134
A.2.3 The final reduce 139
A.3 Acyclic environments 139
A.3.1 Three long environments 141
A.3.2 Two long environments 145

References **151**

Symbols **161**

Index **165**

Samenvatting **171**

Acknowledgements **177**

Curriculum Vitae **179**

CHAPTER 1

Introduction

Robotic manipulation deals with part handling problems in industrial automation such as part feeding, fixturing, loading, assembly and packing, and is therefore of utmost and direct practical importance.

Computational geometry is the field within algorithms and complexity research that deals with geometric objects such as points, lines and surfaces in various dimensions and their interrelationships [10, 48]. This field has evolved into a mature area of research in the late seventies. Computational geometry has become an indispensable tool in the development of algorithms for diverse applications areas like databases, graphics, geographical information systems, and robotics. In robotics, geometric techniques have been applied mostly to the motion planning problem. In this thesis, we apply techniques from computational geometry to robotic manipulation.

The geometric shape of parts is a major parameter in determining the outcome of a physical action on a part, such as a push or grasp by a robot. Hence, computational geometry offers a useful abstraction to investigate these problems. In this thesis, we study planning strategies for orienting parts. We give geometric algorithms for designing various types of devices for orienting parts, aiming to be precise, non-heuristic, and efficient.

In the field of robotic manipulation the goal of the robot, or more precisely, the manipulator, is to ensure desired final states of the objects, or parts, under manipulation; its own final state is not of paramount importance. In contrast to general motion planning (where we plan motions for a mobile robot in a workspace with obstacles [56]), robotic manipulation is designed to take place in a factory or a structured environment; parts typically arrive at a more-or-less regular rate along a conveyer belt. Thus, only limited sensing capabilities are necessary. In fact, Canny and Goldberg [29] propose a RISC approach (Reduced Intricacy in Sensing and Control) to designing manipulation systems for factory environments. Inspired by Whitney's recommendation that industrial robots have simple sensors and actuators [109], they argue that automated planning may be more practical for robot systems with fewer degrees of freedom and simple, fast sensors: light beams instead of cameras, parallel-jaw grippers instead of multi-fingered hands. To be cost-effective, industrial robots should emphasize efficiency and reliability over the potential flexibility of anthropomorphic designs. In addition to these advantages of RISC hardware, RISC systems also lead to positive effects in software: manipulation algorithms that are efficient, robust, and subject to guarantees, more algorithmic and less heuristic-like. Due to these desirable properties, the design of RISC systems and algorithms is a significant branch of applied robotics research. Although at first glance, theoretical research might not appear directly applicable in a real industrial environment, this research opens up a completely new view on robotic manipulation which turns out to be exciting for both theoretical researchers as well as for industrial engineers.

1.1 Automated assembly

The basic tasks in a factory are manufacturing the parts, and combining them into the desired product. In automating these tasks, we want to use robot manipulators that require little or no human intervention. Let us review the various tasks of automated assembly in the light of RISC robotics and computational geometry.
Assembly planning concerns planning the motions of the parts to form the assembly. The general assembly planning problem is so complex that even special cases of it can be intractable [75]. The practical issues are also very complicated and attempting to solve it in all its generality would be attempting to tackle many of the open problems in robotics research all at once! An important subproblem is that of computing an assembly sequence. In practice, most real assemblies can be constructed by a sequence of operations, where each operation merges two rigid subassemblies to make a larger one that stays rigid for the rest of the plan [39, 49]. Theoretical studies point out that even the partitioning problem, in which one wants to separate two bodies, is not easy to solve, and partitioning a planar assembly is NP-complete [54, 97, 112, 113]. This inspires us to begin with low-level tasks and proceed from there.

In the field of robotic manipulation several distinct types of low-level problems exist, most of which at least partially warrant an approach that uses computational geometry. To be able to do so, one is forced to make precise formulations for these problems, while taking into account physical restrictions and properties. We can eventually expect to find enough similar qualities in these precise models that will make it possible to formulate a variety of robotic manipulation problems in a more general geometric framework. This will then allow for the study of generic techniques that can be applied to a variety of different robotic manipulation problems, both in theory and in practice.

One manipulation problem of interest is *fixture planning*. Fixture planning concerns holding an object steady to resist applied wrenches (forces and moments). Fixture planning is a fundamental problem of both the manufacturing and the assembly process, e.g. when one needs to drill a hole in a part, or needs to hold a part when putting it together with another part (or subassembly) [24, 34]. Fixture planning is related to part feeding in the sense that it can be tackled using a RISC approach. In the most basic form, fixture planning tries to immobilize a part using frictionless point contacts.

Over a century ago, Reuleaux [89] considered the problem of constraining motions of objects. Reuleaux proposed a geometric condition (form-closure) for a part to be immobilized using frictionless point contacts, which was later improved by Rimon and Burdick [91, 92]. Recent results report on the existence of fixtures for almost any planar part [63, 72]. Many researchers have found geometric algorithms to find fixtures for a given part [43, 64, 77, 101]. Extensions of fixture planning include algorithms for computing form-closure grasps of two-dimensional and three-dimensional polyhedral objects in the context of point fingers with friction [81–83], and the design of modular fixtures in which the positions of the fingers are restricted to a grid [27, 78, 102, 105–108, 116].

In the assembly process, one often needs to move parts from one pose to another. We need to do so, e.g. when one wants to put multiple parts together, or when one wants to feed a part or a sub-assembly to a robot. Perhaps the most humanoid way of doing this is by grasping the part, and then rotating it or carrying it to another position [4, 23, 26, 63, 77, 86, 88, 99]. Inspired by RISC robotics, many other devices to change the position of parts have been proposed. Among these devices are those that move the part by push operations [3, 60, 61], or by tap operations [52]. It has been researched how the position of parts can be controlled by rolling them between two plates [62]. Recently, it has been shown that applying a carefully controlled vibration to a simple flat plate enables one to very accurately control the pose of multiple independent parts on the plate [90]. It has been investigated what possible poses of a part can be after being dropped on a surface [47, 55].

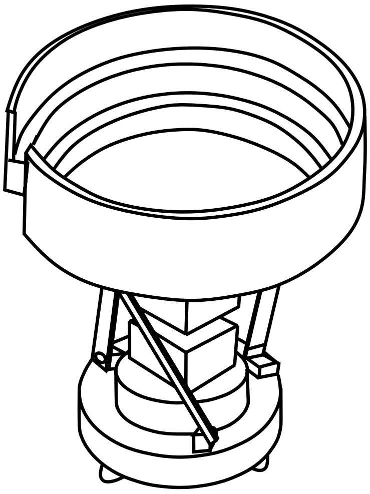


Figure 1.1 A bowl feeder [22].

One step beyond moving a part from a known pose to another (specified) pose is the important manipulation problem of *part feeding*, or *orienting*. Many automated manufacturing processes require parts to be oriented prior to assembly. A part feeder takes in a stream of identical parts in *arbitrary* orientations and outputs them in a uniform orientation. Part feeders often use data obtained from some kind of sensing device to accomplish their task. We consider the problem of *sensorless orientation* of parts for which the initial pose of the part is assumed to be unknown. In sensorless manipulation, parts are positioned and/or oriented using passive mechanical compliance. The input is a description of the part shape and the output is a sequence of open-loop actions that moves a part from an unknown initial pose into a unique final pose. We dedicate the next few paragraphs to an overview of existing sensorless part feeders.

Traditionally, sensorless part feeding is accomplished by the *vibratory bowl feeder*, which is a bowl that is surrounded by a helical metal track and filled with parts [21, 22], see Figure 1.1. The bowl and track undergo an asymmetric helical vibration that causes parts to move up the track, where they encounter a sequence of mechanical devices such as wiper blades, grooves and traps. The majority of these mechanical devices act as filters that serve to reject (force back to the bottom of the bowl) parts in all orientations except for the desired one. Eventually, a stream of parts in a uniform orientation emerges at the top after successfully running the gauntlet. The design of bowl feeders is, in practice, a task of trial and error. It typically takes one month to design a bowl feeder for a specific part [65]. Specific to vibratory bowls, researchers have used simulation [11, 53, 69], heuristics [58], and genetic algorithms [35] to design traps, and developed analysis tools to help designers by rendering the configuration-space for a given combination of part, trap, and obstacle [28]. Sony developed their own part feeding system, which consists of a vibrating tray in which part-specific nests are cut [73, 98]. This feeder operates in batches. During a specified amount of time the tray vibrates to ensure that its holes fill up with parts in the correct orientation. The tray is fed to a robot afterwards.

The disadvantage of the aforementioned feeders is their lack of flexibility to different part shapes. To overcome this problem, researchers have focused on easily adjustable modular part feeding systems. Instead of selecting only parts in the desired orientation by part-specific filters, these modular systems can perform basic actions that *transfer* a part from one orientation into the other. The remaining question is then to design a sequence of basic actions for a given part shape that yields a unique final orientation. In the following, we shall give an overview of flexible part feeders that have been proposed. The question of how to program these part feeders is addressed later.

Many different models and devices have been designed with the goal to automatically orient parts. Sensorless pose planning by transferring orientations has widened up as an area of active research during recent years. We review some results from literature that are close in spirit to this thesis. The early papers on theoretical sensorless orientation report on a tray to manipulate the parts by means of tilting actions [41, 76]. The ability of orienting a small class of polyhedral parts by putting it onto a wobbling table has been investigated [42]. Researchers proposed staircase-like step devices to orient parts that cascade down these steps. The height of the steps, which influences the rotational behavior of the part, is programmable [76, 114].

There is a class of feeders that operate in place. The feeder applies a sequence of basic actions to a part in a designated feeding area. In this case, the parts are first given to the part feeder one by one, then oriented by the feeder, and subsequently further processed by an assembly robot. An important example of an in-place feeder is the parallel jaw gripper which can push and squeeze a part. A grasp action is the combination of orienting the gripper, closing the jaws as far as possible without deforming it, and then opening the jaws. A grasp action causes the orientation of the part to change [26, 32, 45]. The single pushing jaw, which changes the orientation of the part by means of push actions from various directions, has also been advocated as a part feeder [3, 61, 66, 79]. Changing the orientation of a part can also be accomplished by a vibrating surface underneath the part. Vibratory plates and programmable vector fields have been reported as part feeders [20].

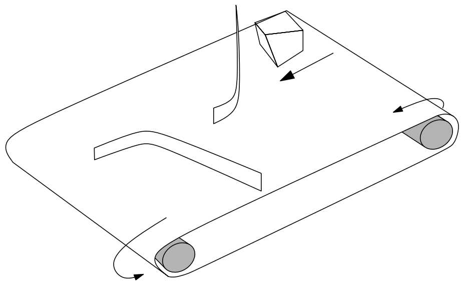


Figure 1.2 Rigid fences over a conveyor.

There is a variety of papers that focus on orienting a stream of parts on a conveyor belt, e.g. by rigid fences that are mounted across the belt. The part is conveyed along the fences, and the motion of the belt effectively turns each slide along a fence into a push action by the fence in the direction opposite to the motion of the belt. The push actions change the orientation of the part [13–15, 80, 110, 111], see Figure 1.2. Inspired by a conveyor belt with rigid fences, the single rotational fence of which the orientation can be controlled by a robot was proposed [2]. The fences over a conveyor and single rotational fence are designed to orient flat parts that lie on the conveyor. Recently, it has been discovered that it is possible to orient a small class of prismatic three-dimensional parts on a conveyor by toppling them over by a sequence of pins that are mounted at various heights above the conveyor [59, 115].

In all forms of pose planning, some motion is induced to the part (by a conveyer belt, a gripper, gravity, or a vibration) so that when it impacts a designed obstacle (fence, another jaw of the gripper, the work table), its pose is affected. The main questions in all these cases is the design of the device and the sequence of operations to be executed to guarantee one (or a small set of) final orientation from an initially unknown orientation. Goldberg [46] considers the *completeness* of some algorithms for part feeders, and, moreover, asks a stronger question: does a solution exist for all instances of a problem? He poses the term *solution-complete* to describe this property of algorithms for part feeders. A part feeder algorithm is solution-complete if for any part, and valid input, the algorithm outputs a design for the feeder that orients any initial orientation of the part to a unique final orientation. Completeness is a desirable property, since it guarantees that an algorithm will behave properly for any valid input. However, solution-completeness is closely related to the way a problem is defined. An algorithm may not be complete for a given problem; however, by

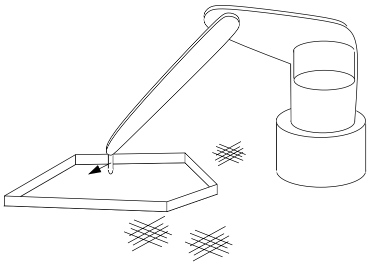


Figure 1.3 Orienting with the pulling finger.

adding additional assumptions limiting the class of inputs, it might be that the algorithm is complete for the resulting modified problem. The larger the class of valid inputs, the better the resulting feeder.

Up to now, many of the approaches to solve the part feeding problem do not consider completeness issues, or when they do, they show completeness for only a simple class of objects. Others are algorithmically poor (for example, requiring exponential time). In this thesis, we aim to theoretical, geometrical solutions to different part feeders. We prove completeness of our algorithms for large classes of parts. In the next section, we present an overview of the results and feeders we address in this thesis.

1.2 Thesis outline

We consider part feeders in the line of thought of the RISC approach of Canny and Goldberg [29]. The feeders in this thesis orient parts without the use of sensors. We carefully embed each of the feeders in a geometric framework. This then allows us to address questions of completeness, and to give efficient algorithms for computing feeder designs.

We consider four different feeders. The first feeder consists of a sequence of fences which are mounted across a conveyor belt. The fences brush the part as it travels down the belt thus reorienting it. The second feeder is a pulling finger which applies a sequence of pull operations that reorients flat parts with elevated edges. The third feeder deals with three-dimensional parts. It is a sequence of (tilted) plates with fences which reorients a three-dimensional polyhedral part as it travels down the plates. The fourth feeder in this thesis is the bowl feeder. We study cut-outs in the track of the bowl to allow only parts in a desired orientation to arrive at the outlet of the bowl.

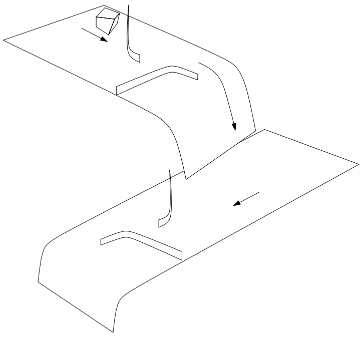


Figure 1.4 Feeding three-dimensional parts with a sequence of plates and fences.

In Chapter 2 we introduce the reader to the area of algorithms for sensorless orientation. We give an overview of work on the mechanics of part feeding that we use in this thesis. We survey known results on part feeding by push actions. Previous work shows that any polygonal part can be oriented by a sequence of push actions unless the part has specific symmetry—it is for example not possible to orient regular polygons. We outline previously known algorithms to compute the shortest sequence of push actions to orient a given part.

In Chapter 3 we discuss part feeding by rigid fences over a conveyor, see Figure 1.2. We present two new algorithms to compute the angles for a fence design. These algorithms are the first algorithms for computing fence designs that require polynomial time. We give new bounds on the complexity of any algorithm to compute sequences of basic (transfer) actions for orienting parts. These results apply to a large class of part feeders. Moreover, we address completeness of fence design. We show that, similar to a sequence of push actions, we can design a sequence of fences to orient any asymmetric polygonal part. We treat modular fence design, in which we restrict ourselves to a discrete set of available fence angles. Also, we incorporate friction in the design of a sequence of fences. A major part of this chapter is published as [15]. Parts of the chapter appeared in [13] and [14].

In Chapter 4 we introduce a new feeder to orient parts: the pulling finger, see Figure 1.3. This feeder can be used to manipulate flat parts with elevated edges in place. The finger is positioned above the part and performs a sequence of pull operations. We identify a basic transfer action for this feeder, and embed it into the larger framework of feeders that orient parts by changing the orientation. We show that for a large class of parts, though smaller than the class of parts that can be oriented by push operations, the feeder can perform its orienting task

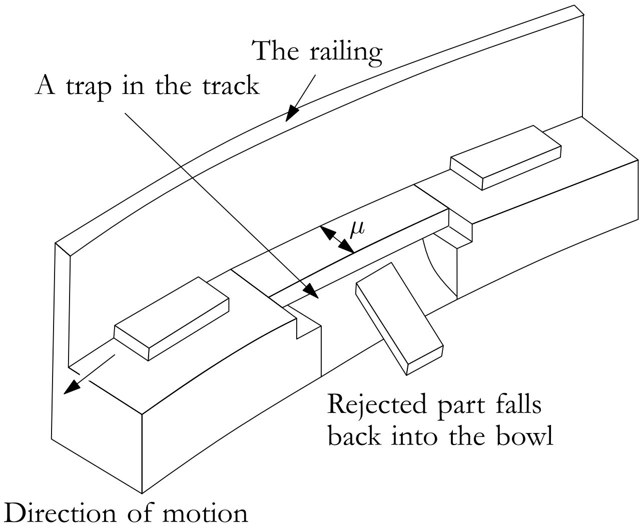


Figure 1.5 Vibratory bowl feeder track [22].

successfully. Moreover, we give an algorithm to compute the shortest sequence of pull operations to orient a given part.

Most efforts in the literature of part feeding concentrate on feeding flat, two-dimensional parts. Efforts to feed three-dimensional parts usually only orient polyhedral parts up to the facet, and not to a unique orientation of the part. In Chapter 5 we show how to orient three-dimensional polyhedral parts. We give the first completeness result for three-dimensional part feeding by a sequence of push operations. We show that any asymmetric polyhedral part can be oriented using push actions by two orthogonal planes. The result is used to show that a feeder which consists of a series of plates and fences can orient any asymmetric polyhedral part, see Figure 1.4. We give an algorithm that computes a sequence of push operations, or similarly, a sequence of plate and fence angles, that orient a given polyhedral part. A (preliminary) version of the work in this chapter appeared in [18], and [19].

Chapter 6 elaborates on feeding parts with the aid of the bowl feeder. An almost infinite amount of features is imaginable to be mounted to the railing of the bowl, and on the feeder track. We shall treat the bowl feeder using a RISC approach. We restrict ourselves to cut-outs in the feeder track. The cut-outs are used to trap, or reject unwanted orientations of the part, thus only keeping the desired orientation, see Figure 1.5. We analyze traps in a geometrical setting. We show how to compute whether a given trap shape will reject a part on the track.

We systematically build a set of adjustable traps in this chapter. We first focus on very simple trap shapes, but also show how more advanced trap shapes can be computed. The work of this chapter appeared partially in [12], [16], and [17]. Each of the chapters is concluded by a discussion of the algorithms and remaining open problems. In Chapter 7 we discuss the work of this thesis in a broader perspective and point out directions for further research.

CHAPTER 2

Pushing

Pushing objects is a regular task. There are numerous places where pushing of objects is used to manipulate them. In automated manufacturing processes, for example, objects are usually transported by a conveyor belt, across which there are various guides to move objects through the system in an appropriate manner. More general forms of pushing often arise, even in every day tasks such as cleaning a table from numerous objects, or moving furniture around the room. In this chapter, we explore push actions to manipulate parts. In Section 2.1 we concentrate on the mechanics of pushing. We discuss previous results that report on the behavior of a part that is pushed by a fence. We show how friction influences the motion of the part. In Section 2.2 we show how to work towards a geometric model of pushing. Within this model, we give an overview of previous work on orienting parts by means of push actions. This chapter is an introduction to part feeding and manipulating. We will use the results of this chapter throughout this thesis.

2.1 Mechanics of pushing

We focus on pushing of parts in order to move them to a desired orientation. We analyze the task of pushing using an approach known as *quasi-static analysis* [66], meaning that we look for a balance among forces which interact with the part, while neglecting inertial forces. This approach can be very accurate with the speeds and scale of objects often encountered in manipulation tasks, but will fail when dynamic forces come into play.

Pushing can be a good way to reduce uncertainty in the state of the task. Mason [66] was the first to analyze the role of pushing in robotic manipulation. In the following, we shall give a short introduction to the role of friction in part manipulation, and then give an overview of important results from the literature on manipulating parts in the plane using pushing. We refer the more interested reader to the work of Mason [65–67] and Peshkin and Sanderson [79] which serve as a basis for this section.

2.1.1 Friction

There are several ways to look at friction. Coulomb was the first to thoroughly research friction in the 18th century [38]. We will use his model of friction, which is rather simple and referred to as *Coulomb friction*. Perhaps the easiest way to explain Coulomb friction is to picture a system with one degree of freedom. Suppose we have a part sliding on an inclined surface. The frictional force is a force opposite to the motion of the part. According to Coulomb's law, the frictional force, f_{friction}, is a fixed fraction of normal force, f_{normal}, acting on the part—the normal force is induced by the gravitational force, f_{grav}, which presses the part onto the surface. This fraction is given by the *coefficient of friction* μ. A simple statement of Coulomb's law for a moving part is:

$$|f_{\text{friction}}| = \mu |f_{\text{normal}}|.$$

If $\mu|f_{\text{normal}}|$ is larger than the resulting force in the direction of the part, then the part will remain at rest. The frictional force now balances the component of the normal force parallel to the surface, and Coulomb's law boils down to

$$|f_{\text{friction}}| \leq \mu |f_{\text{normal}}|.$$

In Figure 2.1(a), we see that the part slides, the frictional force combined with the normal force do not balance the gravitational force ($f_{\text{friction}} + f_{\text{normal}} \neq f_{\text{grav}}$). In Figure 2.1(b), we see a part which remains at rest. The frictional force combined with the normal force cancel out the gravitational force ($f_{\text{friction}} + f_{\text{normal}} = f_{\text{grav}}$). A graphical interpretation of the Coulomb friction is given by the cone with dihedral angle $2 \arctan \mu$, which was probably first presented by Moseley [74]. If the normal force combined with the frictional force (the vector $f_{\text{friction}} + f_{\text{normal}}$) is in the interior of the cone, the part remains stable. Otherwise, the part will slide. It is easy to see whether the combined normal and frictional force is in the cone. If we negate the gravitational force, we check whether this force is in the cone. If this is the case, the part remains stable. The part slides, otherwise.

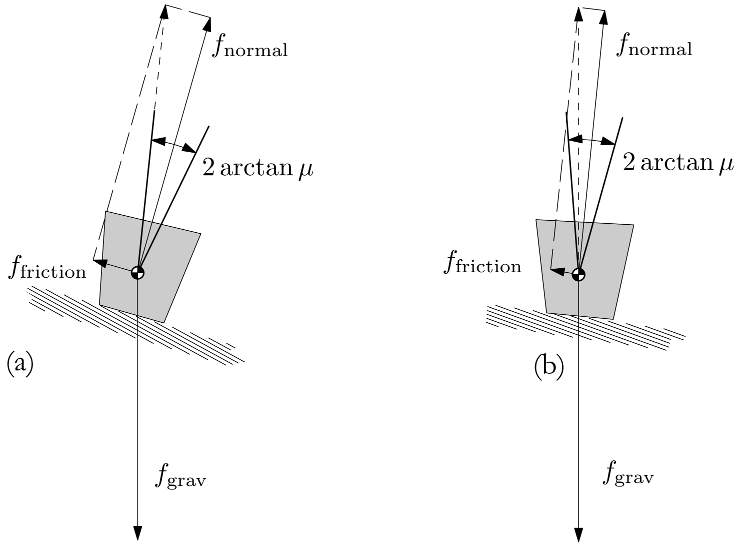


Figure 2.1 A part on an inclined surface. (a) The part slides. (b) The part does not slide, due to friction.

2.1.2 Planar sliding

Many manipulation tasks involve an object sliding on a (flat) planar support surface. In this section, we discuss how a part, which rests on a planar surface behaves when an external force is applied to it. We consider the motion of a part that is pushed by a so called *pushing jaw*. The pushing jaw can be figured as a straight, mobile fence.

Any motion in the plane can modeled by a rotation. A translation can be seen as a rotating around a point at infinity. The exact motion of a part that is pushed by the pushing jaw is dependent on the pressure distribution of the part, and can be found by integrating the forces acting on the part over the surface of the part. Unfortunately, the pressure distribution of the part is, in general, hard to estimate. Nevertheless, there are results in literature which give a bound on position of the center of rotation of a manipulated part [79], or, what turns out to be even more useful, predict the mode of rotation of a manipulated part [68].

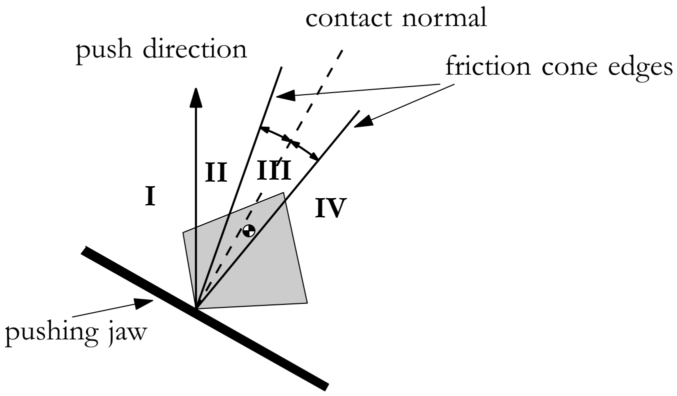


Figure 2.2 The different areas of rotation of the part in vertex-contact with the pushing jaw.

Mode of rotation

Mason [66] describes an easy test to predict the mode of rotation of the part which has a vertex in contact with a pushing jaw. He gives an analysis which boils down to a very simple geometric check. Given the orientation of the part, the coefficient of friction, and the push direction, we can draw three half lines emanating from the contact vertex. One is the line of pushing, which is opposite to the direction of the belt. The two other lines are the boundary edges of the friction cone. The area above the jaw is now divided into four areas, which we enumerate from left to right. This enumeration is independent of the relative order of the push direction and the friction cone edges (see Figure 2.2). The position of the center-of-mass determines the rotation of the part. If it is in areas **I** and **II**, the part will rotate counterclockwisely. If it is in areas **III** and **IV**, the part will rotate clockwisely. If the center-of-mass is on the line separating area **II** from area **III**, the part is in unstable equilibrium, and will neither rotate clockwisely, nor rotate counterclockwisely. The part might either remain in the same position if the separating line is the line of pushing, or slide along jaw otherwise.

We can also predict the mode of rotation if instead of a single vertex, an edge of the part is in contact with the jaw. In this case, we perform the geometric check for the two vertices of the edge, and see if they predict counterclockwise or clockwise rotation. If the vertices disagree, the edge is stable, and the part will not rotate. Otherwise, the edge is unstable and the part will rotate as predicted by both vertices.

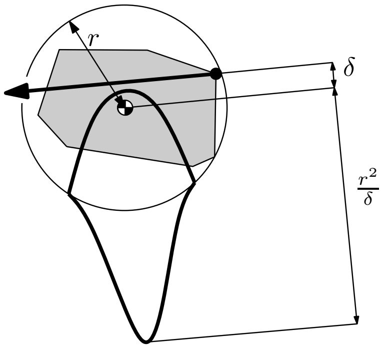


Figure 2.3 The Peshkin bound (thick) on the position of the center-of-rotation for the grey part pushed by the black dot in the direction of the arrow.

Position of center-of-rotation

The exact location of the center-of-rotation of a part which is pushed depends on the exact distribution of friction underneath the part. Since it is very hard to determine this distribution exactly, we settle for an estimation of the position of the (instantaneous) center of rotation. Peshkin and Sanderson [79, 80] described an estimate on this bound, which is referred to as the *Peshkin bound*. Their bound is derived as follows. First, replace the part by a disc with radius r centered at the center-of-mass of the part. Second, estimate all possible pressure distributions of the disc by taking all pressure distributions of only two pressure points. Plotting all possible positions of the instantaneous center of rotation when the part is pushed with the line of pushing at distance δ from the center-of-mass of the part leads to the bound of Figure 2.3. Interesting is the instantaneous center of rotation which gives the slowest rotation. This point is called the *tip point*, and lies on a distance r^2/δ from the center-of-mass of the part.
Strictly speaking, Peshkin and Sanderson only studied dipods: pressure distributions involving only two points of support. They conjectured that the bound they derived is valid for all pressure distributions of the disc. Up till now, no one has been able to prove of disprove this conjecture.

2.2 Geometric modeling

We present geometric algorithms which solve the part feeding problem for various devices. A first step towards designing geometric algorithms for such a practical problem is to appropriately model the physical characteristics of the parts and the manipulator. We consider the different components of a manipulating system: the manipulator, the parts, how the parts are presented to the manipulator, and what needs to be done with the parts. The choice made in selecting each component can affect the overall picture drastically. Therefore, a first step in our research is making smart, but at the same time realistic, choices. These choices can affect the reliability issues of a system considered later.

The shapes of parts involved are of obvious paramount importance in designing manipulator systems and algorithms. The easiest parts to model are parts with constant polygonal cross section. Although many industrial parts do not fall into this category (common CAD primitives in designing parts are piecewise circular, conical or even B-spline arcs), we believe that studying the simple cases first is the best way to build a solid framework for part feeding. In the future, we want to generalize our results to other part shapes.

By a manipulator an arm attached to a robot is meant; in its simplest form, it is a set of joints connected by rigid links. The number of degrees of freedom is a measure of the manipulator's complexity. A simple parallel jaw gripper has only one fine controllable degree of freedom: its orientation. Nevertheless, using the proper algorithms such a manipulator can perform interesting operations.

In order to get a better understanding of the fundamentals of part feeding, literature has focused on simple instances of the part feeding problem. In this section, we introduce the reader to the geometric basics of part feeding by means of push actions.

2.2.1 The push function

In this section we focus on the *push function* of a part. Let P be a polygonal part with n vertices and center-of-mass c. We assume that a fixed coordinate frame is attached to P. Directions are expressed relative to this frame. The contact direction of a supporting line ℓ of P is uniquely defined as the direction of the normal of ℓ pointing into P.

Definition 2.1 [66] *The radius function* $\delta : [0, 2\pi) \rightarrow \mathbb{R}^+$ *maps a direction* ϕ *onto the distance from* c *to the supporting line of* P *with contact direction* ϕ.

For a polygonal part, the radius function is a continuous piecewise sinusoidal function, and can be computed in $O(n)$ time by checking each vertex [66]. We shall show that the final orientation of a part that is being pushed can be determined from its radius function. First, we define the notion of *convexity* and the *convex hull* of subsets of the plane.

Definition 2.2 *A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$ the line segment (p, q) is completely contained in S.*

Definition 2.3 *The convex hull, $\mathcal{CH}(S)$, of a set S is the smallest convex set that contains S.*

Since the pushing jaw touches P at its convex hull, we assume without loss of generality that P is convex. In most cases, parts will start to rotate when pushed. If pushing in a certain direction does *not* cause the part to rotate, then the contact normal at one of its points of contact with the jaw passes through the center-of-mass [66]. We refer to the corresponding direction of the contact normal as an *equilibrium* push direction or orientation. If pushing does change the orientation, then this rotation changes the orientation of the pushing device relative to the part. We assume that pushing continues until the part stops rotating and settles in a stable equilibrium pose. A stable equilibrium pose corresponds to a (local) minimum of the radius function.

The push function $\varpi : [0, 2\pi) \to [0, 2\pi)$ links every orientation ϕ to the orientation $\varpi(\phi)$ in which the part P settles after being pushed by a jaw with initial contact direction ϕ (relative to the frame attached to P) [26, 45]. The rotation of the part due to pushing causes the contact direction of the jaw to change. The final orientation $\varpi(\phi)$ of the part is the contact direction of the jaw after the part has settled. The equilibrium push directions are the fixed points of the push function ϖ.

The push function ϖ of a polygonal part consists of steps, which are open intervals $I \subset [0, 2\pi)$ for which $\varpi(\phi) = \sigma$ for all $\phi \in I$, and some constant $\sigma \in I$. The steps of the push function are easily constructed from the radius function δ. If the part is pushed in a direction corresponding to a point of non-horizontal tangency of the radius function then the part will rotate in the direction in which the radius decreases. The part finally settles in an orientation corresponding to a local minimum of the radius function. As a result, all points in the open interval I bounded by two consecutive local maxima of the radius function δ map onto the orientation $\phi \in I$ corresponding to the unique local minimum of δ on I. Note that ϕ itself maps onto ϕ because it is a point of horizontal tangency. This results in the steps of the push function. We define two open intervals $l(\sigma) = \{\phi < \sigma | \varpi(\phi) = \sigma\}$ and $r(\sigma) = \{\phi > \sigma | \varpi(\phi) = \sigma\}$ for each fixed point—or equilibrium orientation—σ of the push function. We refer to these intervals as σ's left and right environment respectively. The interval $l(\sigma)$ corresponds to the half-step left of $\sigma = \varpi(\sigma)$ and $r(\sigma)$ corresponds to the half-step right of $\sigma = \varpi(\sigma)$ (see Figure 2.4). Note that an equilibrium σ is stable if and only if $l(\sigma) \neq \varnothing$, and $r(\sigma) \neq \varnothing$. We denote by $\Sigma = \sigma_1, \ldots, \sigma_{|\Sigma|}$ the cyclic sequence of stable equilibria, cut at some arbitrary orientation, and ordered by increasing angle. Besides the steps, there are isolated points satisfying $\varpi(\phi) = \phi$ in the push function, corresponding to local maxima of the radius function. Figure 2.4 shows a polygonal part and its push function. The final orientation of the part while being pushed is described by the push function. From Theorem 2.8 (which will be presented shortly) it follows that any step function having only non-zero-length half-steps represents a polygonal part, and is therefore a valid push function. We use the abbreviation ϖ_α to denote the (shifted) push function defined by

$$\varpi_\alpha(\phi) = \varpi((\phi + \alpha) \bmod 2\pi),$$

for all $\phi \in [0, 2\pi)$. The shifted push function $\varpi_\alpha(\phi)$ corresponds to the final orientation of a part in initial orientation ϕ when we first reorient the jaw by α, and subsequently apply the jaw. Such a reorientation and application of the jaw is referred to as a *basic push action*.

The *period* of a push function ϖ is the smallest positive d for which $\varpi(\phi) = \varpi((\phi + d) \bmod d)$ for all $\phi \in [0, 2\pi)$. Any part can only be oriented up to *(periodic) symmetry* in its push function.

2.2.2 Orienting planar parts by pushing

Natarajan [76] first formalized the problem of part orienting in terms of a set of functions–called *transfer functions*–which map orientations of parts to orientations of parts. The class of transfer functions permitted in a plan would be determined by the possible (basic) actions of the hardware used to design the part feeder. Consider a part feeding system that accepts as input a set of part orientations Σ. Based on a definition by Akella *et al* [2], we might say that a system has the *feeding property* if there exists some orientation σ^*, usually in Σ, such that the system outputs parts only in orientation σ^*.

A sequence $\sigma_1, \ldots, \sigma_n$ of elements of a set Σ is *ordered* if $\sigma_1, \ldots, \sigma_n$ are encountered in order when the generating cycle of Σ is traced *once*, starting from σ_1. A function $\varpi : \Sigma \rightarrow \Sigma$ is *monotonic* if for any ordered sequence $\sigma_1, \ldots, \sigma_n$ the sequence $\varpi(\sigma_1), \ldots, \varpi(\sigma_n)$ is also ordered. We can consider the set Σ as the set of states of a part, i.e. the set of possible orientations of the part. The

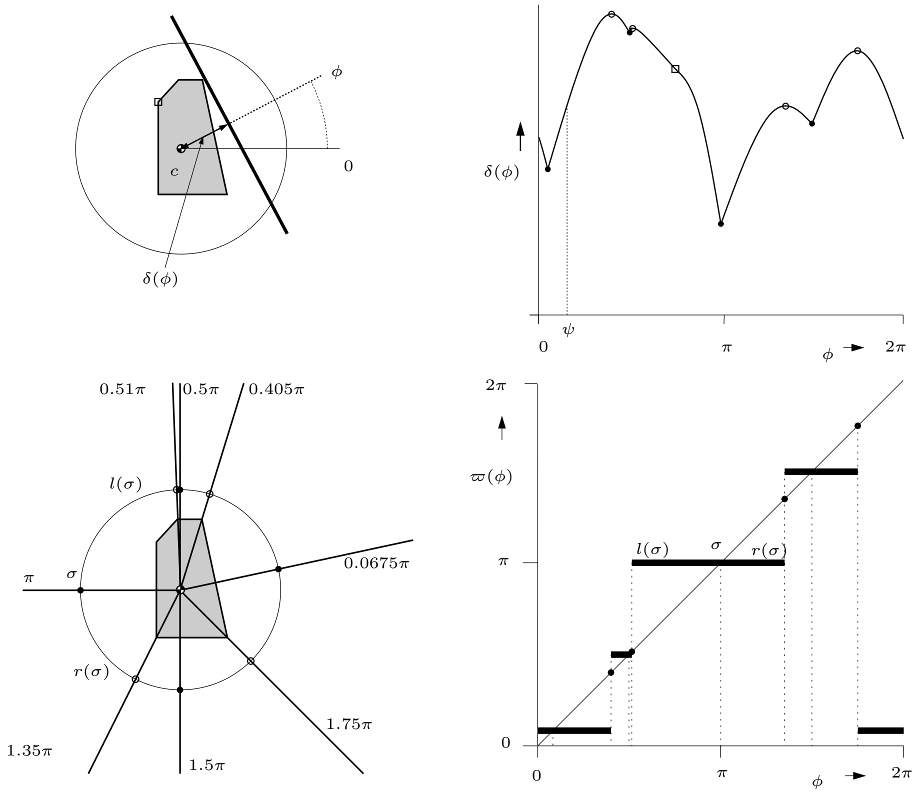


Figure 2.4 A polygonal part and its radius and push function. The minima of the radius function correspond to normals to polygon edges that intersect the center-of-mass. The maxima correspond to tangents to polygon vertices whose normals intersect the center-of-mass. The horizontal steps of the push function are angular intervals between two successive maxima of the radius function.

transfer function of the part feeder maps a basic action of the feeder onto the set of resulting orientations after application of the basic action, e.g. the set of orientations of a part after a push action of a jaw.

Natarajan [76] proved that for an automated parts orienter with monotonic transfer functions, there exists a polynomial algorithm to design a sequence of basic actions that has the feeding property. Eppstein [40] improves the running time of Natarajan's algorithm. He shows how to compute the optimal sequence of basic actions (in terms of the length of the sequence) for a part P with n possible orientations for which m distinct basic actions exist in $O(mn^2)$ time, if such a sequence exists.

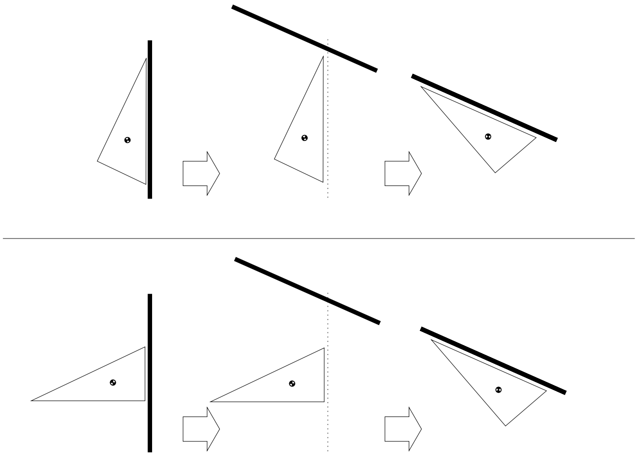


Figure 2.5 Colliding two different orientations of the same part by a single basic push action, i.e. reorienting the jaw by the same angle, and subsequently applying the jaw.

In the previous section, we introduced the push function, which is a transfer function. The next step is to show how to use push operations to orient a part. We define a *push plan*.

Definition 2.4 *A push plan is a sequence* $\alpha_1, \ldots, \alpha_k$ *such that* $\varpi_{\alpha_k} \circ \cdots \circ \varpi_{\alpha_1}(\phi) = \sigma^*$ *for all* $\phi \in [0, 2\pi)$ *and some fixed* $\sigma^* \in [0, 2\pi)$.

In words, a push plan is a sequence of push operations that has the feeding property. Algorithms that compute push plans, are all based on colliding multiple possible orientations of P by means of a basic push action. In Figure 2.5 we see the same basic action of the pushing jaw for two different orientations of the same part. The basic action collides the two orientations.

It is not hard to see that the push function is monotonic. Moreover, we can bound the number of different push directions by $O(n^2)$—each of the $O(n)$ possible orientations of P can be mapped onto the preimage of any other orientation. Hence, we can compute the shortest push plan that has the feeding property in $O(n^4)$ time, using the algorithm of Eppstein.

A better understanding of the push function leads to more efficient algorithms. Goldberg [45] showed that any polygonal part can be oriented up to the periodic symmetry of the push function by a sequence of pushes. Chen and Ierardi [32] constructively proved that any polygonal part with n vertices can be oriented by $O(n)$ pushes. They showed that this bound is tight by constructing n-gons that require $\Omega(n)$ pushes to be oriented. Chen and Ierardi use a sequence of equivalent basic actions to orient a polygonal part with a unique longest right environment of length λ. Let P be such a polygon and let σ^* be such that $|r(\sigma^*)| = \lambda$. Each basic action consists of a reorientation of the jaw by an angle of $\lambda - \epsilon$, with $\epsilon > 0$ such that $\lambda - \epsilon > |r(\sigma)|$ for any equilibrium orientation $\sigma \neq \sigma^*$, and a subsequent push. Every basic action puts the part into an equilibrium orientation. After each basic action, the part is therefore in one of a finite number of equilibrium orientations. Let us label the $|\Sigma|$ equilibrium orientations $\sigma_1, \ldots, \sigma_{|\Sigma|}$ in order of *increasing* angle ending with $\sigma_{|\Sigma|} = \sigma^*$. After the first push, the part P can be in any of the equilibrium orientations $\sigma_1, \ldots, \sigma_{|\Sigma|}$. Chen and Ierardi show that every next basic action eliminates the first orientation in the sequence as possible orientation of the part. Assume that P is in one of the orientations $\sigma_i, \ldots, \sigma_{|\Sigma|}$, for some $i \leq |\Sigma|$. The key idea behind the proof is that a next basic action will cause P, when in orientation $\sigma_{|\Sigma|}$, to stay in orientation $\sigma_{|\Sigma|}$ because $\lambda - \epsilon < |r(\sigma_{|\Sigma|})| = |r(\sigma)|$, and, when in orientation $\sigma_{\bar{\imath}}$ for some $i \leq \bar{\imath} \leq |\Sigma|$, to move into some orientation $\sigma_{\bar{\imath}'}$ with $i + 1 \leq \bar{\imath}' \leq |\Sigma|$ because $\lambda - \epsilon > |r(\sigma_{\bar{\imath}})|$. Upon completion of the basic action, the part will, therefore, be in one of the orientations $\sigma_{i+1}, \ldots, \sigma_{|\Sigma|}$. As a consequence, a total of $|\Sigma| + 1$ basic actions suffices to put P into orientation $\sigma_\Sigma = \sigma^*$. In other words, the sequence $(\lambda - \epsilon)^{|\Sigma|+1}$ is a valid push plan for P. In a similar manner, we could use reorientations by $-(\reflectbox{\ensuremath{\lambda}} - \epsilon)$, and obtain the sequence $(-\reflectbox{\ensuremath{\lambda}} + \epsilon)^{|\Sigma|+1}$, where $\reflectbox{\ensuremath{\lambda}}$ is the length of the unique longest left environment.

If there is no unique largest right, or left environment, Chen and Ierardi apply their so-called *Stretching Lemma* which in general uses any reorientation of the pushing jaw. The Stretching Lemma shows that we can shift two possible orientations which both have a maximal left or right environment closer to each other with one single push; in other words: we can break the symmetry if there are multiple orientations with equally long maximal left or right environments by means of a single basic action. The following lemma from the paper of Chen and Ierardi shows us that any interval of possible orientations can be mapped onto a shorter interval of possible orientations by a single push.

Lemma 2.5 [32] *Let* $0 < d < 2\pi$. *If* ϖ *is not periodic with period d, then there are orientations* ϕ, ψ *such that* $d = |(\phi, \psi)|$, $\phi \in l(\sigma_i)$, $\psi \in r(\sigma_j)$ *for some orientations* σ_i, σ_j, *and consequently* $d > |(\varpi(\phi), \varpi(\psi))|$.

From Lemma 2.5 follows Chen and Ierardi's Stretching Lemma. We only sketch the proof. The interested reader is referred to Chen and Ierardi's paper.

Lemma 2.6 Stretching Lemma [32] *Let* P *be a polygonal part, and* ϖ *its push function. Suppose that* ϖ *is aperiodic, and let the possible orientation of the part be one of* $\sigma_i, \ldots, \sigma_{|\Sigma|}$ *for some* $i > 1$. *If* $|r(\sigma_i)| = |r(\sigma_{|\Sigma|})|$, *then there are always two basic push actions such that afterwards the possible orientation of* P *is in* $\sigma_{i+1}, \ldots, \sigma_{|\Sigma|}$.

Proof: (Sketch) The orientation of P is in the interval $[\sigma_i, \sigma_{|\Sigma|}]$. We choose the reorientation of the first basic action such that σ_i, the first orientation, is mapped onto a left environment, and $\sigma_{|\Sigma|}$, the last orientation, onto a right environment. This reorientation exists according to Lemma 2.5. The distance between the first and last orientation is reduced by the first basic action. It is not hard to see that we can find a second basic action which maps the last orientation back onto $\sigma_{|\Sigma|}$ and the first orientation onto an orientation in $[\sigma_{i+1}, \sigma_{|\Sigma|}]$. From the monotonicity of the push function follows that the orientation of P after the two basic actions is in $[\sigma_{i+1}, \sigma_{|\Sigma|}]$. □

The asymptotic length of the resulting plans remains the same for this class of parts. The observation that $|\Sigma| = O(n)$ leads to the following result.

Lemma 2.7 [32] *A polygonal part* P *with* n *vertices can be oriented up to periodic symmetry by* $O(n)$ *push operations.*

Goldberg [45] gave a greedy algorithm for computing the shortest push plan for a polygon. His algorithm runs in $O(n^2)$ time, which is asymptotically better than Eppstein's general algorithm. For a parallel jaw gripper, whose basic actions reorient the part by squeezing, similar bounds are derived. Van der Stappen *et al.* [100] show that for long and thin parts, the number of push or squeeze operations which are needed to orient P can be reduced to $O(1)$.

2.2.3 From a push function to a part

In this subsection we show that, given a step function, there is a polygonal part which has this step function as the push function. This shows that the study of push functions is equivalent to the study of pushing polygonal parts. In [85], Rao and Goldberg gave related results for diameter functions, which predict the rotation of a part, when squeezed by a parallel jaw gripper.
Let ϖ be a push function. Let γ be the minimum length of all left and right environments. If $\gamma > 0$, then we can construct a part with push function ϖ. It is easily verified that this constraint is satisfied by any push function of a

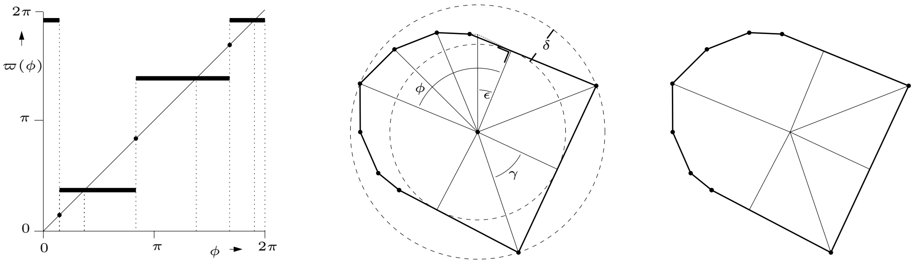


Figure 2.6 The construction of a polygonal part for the given push function on the left. For one environment of the part, $m = \phi/\epsilon$ regions bounded by rays with angular distance ϵ are depicted, together with the resulting polyline.

polygonal part without meta-stable edges, i.e. the perpendicular projection of the center-of-mass on an edge does not coincide with a vertex of the edge.

We construct a polygon for which every (local) maximum in the radius function is a radius of length 1 and every (local) minimum is a radius of length $\max\{0.5, \cos\gamma\}$. The resulting polygon must be convex, and the only minima and maxima of its radius function must be the aforementioned extrema: each stable equilibrium orientation in the push function must correspond to an edge of the polygon such that the perpendicular projection of the center-of-mass lies between the endpoints of the edge. Furthermore, the direction of the projection equals the equilibrium orientation. The isolated equilibrium points in the push function must correspond to vertices of the polygon that lie at a distance 1 from the center-of-mass such that there is a tangent to the vertex whose normal passes through the center-of-mass. So, if the push function has $|\Sigma|$ stable equilibria, we obtain an alternating collection of $|\Sigma|$ vertices at distance 1 from the center-of-mass and $|\Sigma|$ edges at distance $\max\{0.5, \cos\gamma\}$.

It remains to connect the vertices to the edges. These connecting chains should create no additional minima or maxima in the radius function. So, we must make sure that the radius is strictly decreasing from the vertex to the adjoining edges. To this end we start in each minimum a polyline that spirals out to the maximum to the left and to the right. We make sure that no segment of the polyline connecting a minimum to a maximum has an orthogonal line crossing the center-of-mass of the part.

We construct a polyline from a minimum orientation σ^- to an adjoining maximum orientation σ^+ as follows. Without loss of generality, we assume that the interval between σ^- and σ^+ is the right environment of σ^-. We divide the angular interval (σ^-, σ^+) into m equal sized intervals. Let $\phi = |(\sigma^-, \sigma^+)|$, then $\epsilon = \frac{\phi}{m}$ is the size the intervals between σ^- and σ^+. The vertices of the polyline are placed on rays emanating from c with angle $\sigma^-, \sigma^- + \epsilon, \ldots, \sigma^+$. Let $\delta > 0$ denote the difference in radius of the maxima and the minima of the part. The i-th vertex of the polyline has distance $1 - (1 - \frac{i}{m}) \cdot \delta$ to c (See Figure 2.6 for an example of a construction of a part from a push function).

In order for the part to have no minimum or maximum in the angular interval (σ^-, σ^+), we must choose ϵ small enough to satisfy the following constraint: for every pair of successive vertices σ and σ' on the polyline, with distance r respectively $r + \frac{\delta}{m}$ to the center-of-mass of the part, the vertical projection of c on the supporting line of (σ, σ') lies not on the segment. Thus, the vertex σ' is more distant from c than the intersection of the line through σ, which is orthogonal to the ray of σ, and the ray of σ'. Using trigonometry this constraint is given by

$$\frac{r}{r + \frac{\delta}{m}} = \frac{r}{r + \epsilon\frac{\delta}{\phi}} < \cos\epsilon.$$

Using a Taylor expansion gives us a lower bound for $\cos\epsilon$:

$$\frac{r}{r + \epsilon\frac{\delta}{\phi}} < 1 - \frac{1}{2}\epsilon^2,$$

and after some manipulation, using $r < 1$,

$$0 < \frac{\delta}{\phi}\epsilon - \frac{1}{2}\epsilon^2 - \frac{\delta}{2\phi}\epsilon^3.$$

Since $\frac{\delta}{\phi}$ is a constant, we can choose ϵ small enough to satisfy this constraint, and then there is no contact normal of the part in the angular interval (σ^-, σ^+) intersecting c. To make σ^- a minimum, we extend the normal of the circle with radius $1 - \delta$ from σ^- to the intersection with the constructed polyline, and cut away the area of the part outside the extension. We conclude that we can connect each minimum to its successive maximum, and thus construct a convex polygonal part for every push function with non-zero minimum length of left and right environments.

Theorem 2.8 *Let ϖ be a step function with only non-zero-length half-steps. There exists a polygonal part P with its center-of-mass in the interior that has push function ϖ.*

It is not hard to see that if a push function has only zero-length left environments, then it is not possible to construct a convex part for this push function. Hence, there exist push functions with zero-length half-steps that do not correspond any part.

CHAPTER 3

Fence design

The problem of *fence design* is to determine a sequence of fence orientations, such that the fences with these orientations align a part as it moves down a conveyor belt and slides along these fences [25, 80, 111]. In Figure 3.1, the reader can find an example of a fence design. The motion of the belt effectively turns each slide into a push action by the fence in the direction normal to the fence. The fact that the direction of the push, must have a non-zero component in the direction opposite to the motion of the belt imposes a restriction on successive push directions. Fence design can be regarded as finding a constrained sequence of push directions (see Subsection 3.1 for the actual constraints). The additional constraints make fence design considerably more difficult than sensorless orientation by a pushing jaw.

Wiegley *et al.* [111] gave an exponential algorithm for computing the shortest sequence of fences for a given part, if such a sequence exists. They conjectured that a fence design exists for any polygonal part. In this chapter, we prove the conjecture that a fence design exists for any polygonal part. In addition, we give an $O(n^3)$ algorithm for computing a fence design of minimal length (in terms of the number of fences used). This algorithm improves the general algorithm for part feeding problems with monotonic transfer functions due to Eppstein [40], which runs in $O(n^4)$ time. The algorithm is easy to implement and the resulting program returns fence designs for input parts within a fraction of a second. The program can be tuned to take into account certain quality requirements on the fence design, like minimum and maximum (successive) fence angles to prevent impractical steep and shallow fences and a long series of fences on a single side of the belt, which would require an impractically wide conveyor belt. The

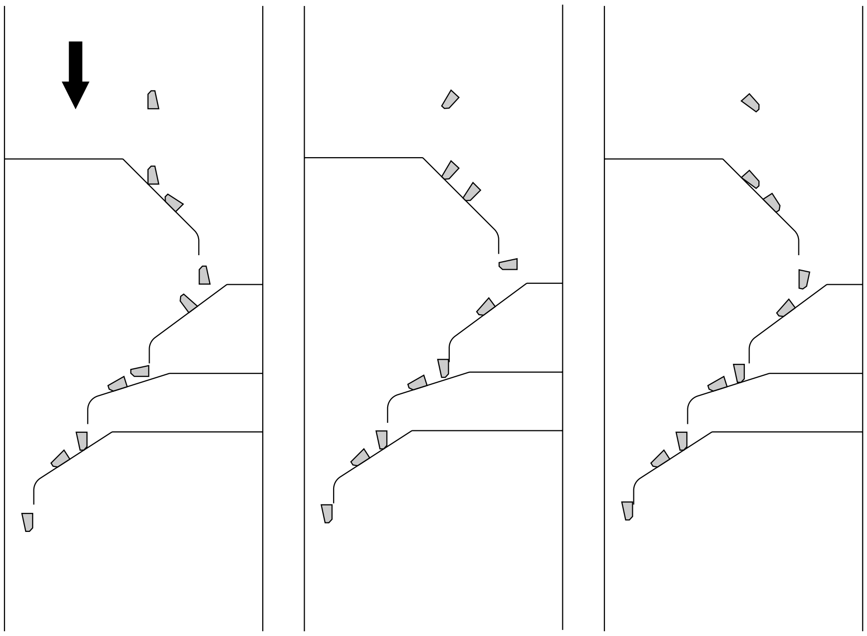


Figure 3.1 Three overhead views of the same conveyor belt and fence design. The traversals for three different initial orientations of the same part are displayed. The traversals show that the part ends up in the same orientation in each of the three cases.

incorporation of quality measures increases the running time of the algorithm to $O(n^4)$. We show that fence designs of length $O(n)$ exist for a large class of parts. The algorithm can be modified to take into account friction between the part and the fences. Also, we can change the algorithm to compute fence designs which use fences with orientations from a given set of allowed orientations. We refer to such fence designs as *modular fence designs*. The running time of the algorithm is then $O(mn^2)$, where m is the number of available fence angles. We give an output-sensitive algorithm which computes the shortest fence design. This algorithm runs in time $O(kn \log n)$, with k the number of fences in the design. We can adopt this output-sensitive algorithm to compute modular fence designs of fence designs with friction in similar time bounds.

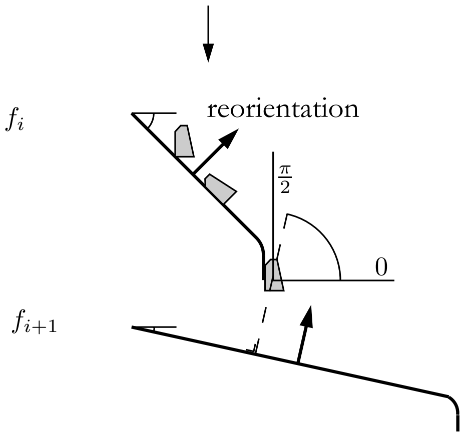
Throughout this section, we assume zero friction between the part and the fences, unless stated otherwise. Since any push action acts on the convex hull of the part rather than on the part itself, we assume without loss of generality that the part under consideration is convex. Furthermore, we assume that the parts do not have meta-stable edges.

This chapter is organized as follows. We introduce the reader to fence design by presenting the geometric model of fence designs in Section 3.1. In that section, we also show the relation between fence design and the more general push plans. The remainder of the chapter contains three major parts: we give different algorithms for computing fence designs, we elaborate on the existence of fence designs for polygonal parts, and we extend fence design to fences with friction and modular fences. In Section 3.2 we give the basic algorithm to compute fence designs. In Section 3.3 we show how to compute fence designs incrementally. In Section 3.4 we consider completeness, i.e., we show that most parts can be oriented up to symmetry by a fence design. For particular parts with a specific kind of symmetry in the push function, this proof of completeness is more difficult. We dedicate Section 3.5 to these parts. Section 3.6 deals with fence designs with friction. Section 3.7 treats modular fence designs. We conclude this chapter in Section 3.9, and pose several open problems.

3.1 Geometric modeling

We address the problem of designing a shortest possible sequence of fences $f_1, \ldots, f_k$ that will orient P when it moves down a conveyor belt and slides along these fences. Let us assume that the conveyor belt moves vertically from top to bottom, as indicated in the overhead view in Figure 3.1. We distinguish between left fences, which are placed along the left belt side, and right fences, which are placed along the right side. The fence angle of a fence f_i denotes the angle between the upward pointing vector opposing the motion of the belt and the normal to the fence with a positive component in upward direction. The motion of the belt turns the sliding of the part along a fence into a push by the fence. The direction of the push is—by the zero friction assumption—orthogonal to the fence with a positive component in the direction opposing the motion of the belt. Thus, the motion of the belt causes any push direction to have a positive component in the direction opposing the belt motion. We now transform this constraint on the push direction relative to the belt into a constraint on successive push directions relative to the part.

Brokowski *et al.* [25] use the Peshkin bound [79, 80] on the position of the center-of-rotation of a part that is pushed by a fence as a basis to design a curved tip to fences across a conveyor belt. Sliding along a fence f_i causes one of P's edges, say e, to align with the fence. The curved tip of the fence guarantees that e is aligned with the belt sides as P leaves the fence. If f_i is a left fence then e faces the left belt side (see Figure 3.2). If f_i is a right fence, it faces the right side.



(a)

t_i	α_{i+1}	t_{i+1}
left	$(0, \pi/2)$	left
left	$(\pi/2, \pi)$	right
right	$(-\pi, -\pi/2)$	left
right	$(-\pi/2, 0)$	right

(b)

Figure 3.2 (a) For two successive left fences, the reorientation of the push direction lies in the range $(0, \pi/2)$. (b) The ranges op possible reorientations of the push direction for all pairs of fence types.

Assume f_i is a left fence. The reorientation of the push direction is the difference between the final contact direction of f_i and the initial contact direction of f_{i+1}. At the moment of leaving f_i, the contact direction of f_i is perpendicular to the belt direction and towards the right belt side. So, the reorientation of the push direction is expressed relative to this direction.

Figure 3.2(a) shows that the reorientation α_{i+1} is in the range $(0, \pi/2)$ if we choose f_{i+1} to be a left fence. If we take a right fence f_{i+1} then the reorientation is in the range $(\pi/2, \pi)$. A similar analysis can be done when P leaves a right fence and edge e faces the left belt side. The results are given in the table of Figure 3.2(b).

The table shows that the type t_i of fence f_i imposes a bound on the reorientation α_{i+1}. Application of the same analysis to fences f_{i-1} and f_i and reorientation α_i leads to the following definition of a valid fence design [111].

Definition 3.1 [111] *A fence design is a push plan* $\alpha_1, \ldots, \alpha_k$ *satisfying for all* $1 \leq i < k$*:*

- $\alpha_i \in (0, \pi/2) \cup (-\pi, -\pi/2) \Rightarrow \alpha_{i+1} \in (0, \pi/2) \cup (\pi/2, \pi)$
- $\alpha_i \in (-\pi/2, 0) \cup (\pi/2, \pi) \Rightarrow \alpha_{i+1} \in (-\pi/2, 0) \cup (-\pi, -\pi/2)$.

The push plan on the left in Figure 3.3 satisfies the constraints of Definition 3.1, and is therefore also a fence design.

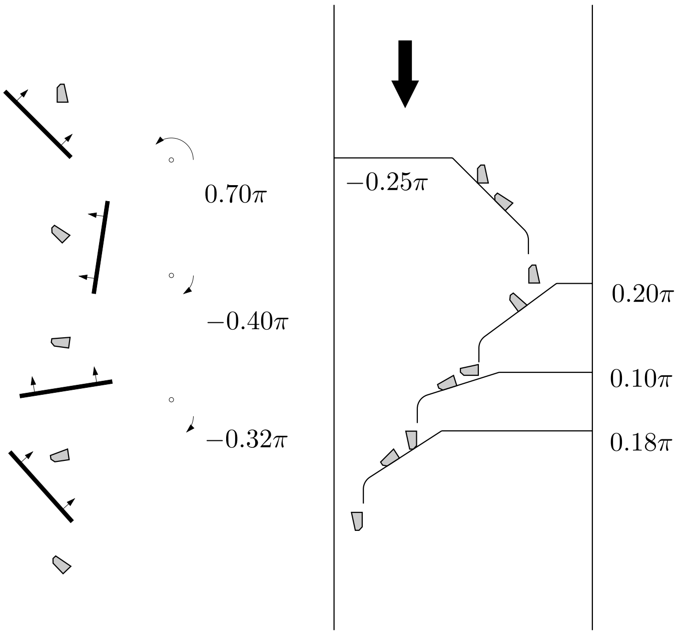


Figure 3.3 The left picture shows a plan for a pushing jaw. The jaw reorientations are (from top to bottom) 0.70π, -0.40π, and -0.32π. The conveyor belt on the right orients the same part. The fence angles, which are measured relative to the upward vector opposing the direction of belt motion, are -0.25π, 0.20π, 0.10π, and 0.18π. (Note that the sequence of fence orientations is not the same as the sequence of orientations of the pushing jaw because the curved fence tip rotates the stable edge of the part to become aligned with the belt direction.)

3.2 Computing fence designs

We denote the sequence of *stable* equilibrium orientations of P by Σ. As every fence puts the part in a stable equilibrium orientation, the part is in one of these $|\Sigma| = O(n)$ orientations as it travels from one fence to another. Let us label these stable equilibria $\sigma_1, \ldots, \sigma_{|\Sigma|}$. After the first fence, a part can be in any of the stable equilibria. The problem is to reduce the set of possible orientations of P to one stable equilibrium σ^* by a sequence of fences. We build a directed graph on all possible *states* of the part as it travels from one fence to a next fence. A state consists of a set of possible orientations of the part plus the type (left or right) of the last fence, as the latter imposes a restriction on the reorientation of the push direction. Although there are $2^{|\Sigma|}$ subsets of $\{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$, we shall see that we can restrict ourselves to subsets consisting of sequences of adjacent stable equilibria. Any such sequence can be represented by the closed interval I

defined by the first and last stable orientation σ_i and σ_j, $i, j \in \{1, \ldots, |\Sigma|\}$ of the sequence. The resulting graph has $O(|\Sigma|^2)$ nodes.

Consider two graph nodes (I, t) and (I', t'), where $I = [\sigma_i, \sigma_j]$ and I' are intervals of stable equilibria and t and t' are fence types. Let $\mathcal{I}_{t,t'}$ be the open interval (of length $\pi/2$) of reorientations admitted by the successive fences of types t and t' according to the table in Figure 3.2(b). There is a directed edge from (I, t) to (I', t') if there is an angle $\alpha \in \mathcal{I}_{t,t'}$ such that a reorientation of the push direction by α followed by a push moves any stable equilibrium in I into a stable orientation in I'. To check this condition, we determine the preimage $(\phi, \psi) \supseteq I'$ of I' under the (monotonic) push function. Observe that if $|I| = \sigma_j - \sigma_i < \psi - \phi$, any reorientation in the open interval $(\phi - \sigma_i, \psi - \sigma_j)$ followed by a push will map I into I'. We add an edge from (I, t) to (I', t') if the intersection of $(\phi - \sigma_i, \psi - \sigma_j)$ and the interval $\mathcal{I}_{t,t'}$ of admitted reorientations is non-empty, and label this edge with $(\phi - \sigma_i, \psi - \sigma_j) \cap \mathcal{I}_{t,t'}$. For convenience, we add a source and a sink to the graph. We connect the source to every node (I, t) in the graph for which I contains all $|\Sigma|$ stable equilibria, and we connect every node (I, t) with $I = [\sigma_i, \sigma_i]$ to the sink. Every path from the source to the sink now represents a fence design. A fence design of minimum length corresponds to a shortest such path.

Lemma 3.2 *The shortest path in the graph corresponds to the shortest fence design, if a fence design exists.*

Proof: We prove that each path corresponds to a fence design and vice versa. The theorem follows immediately if we realize that every edge in the path corresponds to a fence in the design.

($\Rightarrow$) If there is a path, we must prove that there is a corresponding fence design. Since there is an edge from (I, t) to (I', t') if and only if the successive fences t and t' allow for a reorientation that maps I into I', this follows immediately.

($\Leftarrow$) If there is a fence design, we prove there is a path in the graph that represents this fence design. Let $f_1, \ldots, f_k$ be a fence design. We track the possible orientations of the fence design, and prove by induction that for every set of possible orientations there is a node in the graph. Furthermore, we prove that there is a path from the source to such a node. Let Σ_i denote the set of all possible orientations of P leaving f_i. It is easy to see that for each Σ_i there are multiple nodes that include the set of possible orientations.

After the first fence f_1, all stable equilibria are possible. Since we added edges from the source to all nodes containing all stable orientations, these nodes are reachable.

We now assume that for fence f_i having type t in our fence design the nodes (I, t) with $I \supseteq \Sigma_i$ are reachable from the source. Let t' be the fence type of f_{i+1}. Let (I', t') be a node such that $I' \supseteq \Sigma_{i+1}$. Let (ϕ, ψ) denote the preimage of I'. Since the push function is monotonic and by existence of the fence design which maps Σ_i onto Σ_{i+1}, there is an admitted reorientation α_{i+1} by f_{i+1} such that $(\phi - \alpha_{i+1}, \psi - \alpha_{i+1}) \supset \Sigma_i$. Therefore, let (I, t) be a node such that $(\phi - \alpha_{i+1}, \psi - \alpha_{i+1}) \supset I \supseteq \Sigma_i$. There is an edge from (I, t) to (I', t'), and there is a path from the source to (I, t). Since $I' \supseteq \Sigma_{i+1}$ is arbitrary, all (I', t') with $I' \supseteq \Sigma_{i+1}$ are reachable from the source. □

An important observation is that some graph edges are redundant if we are just interested in a fence design of minimum length. Consider a node (I, t) and all its outgoing edges to nodes $(I' = [\sigma_i, \sigma_j], t')$ for a fixed left endpoint σ_i and a fixed fence type t'. We show that only one of these outgoing edges is sufficient. The following lemma is the key to this result.

Lemma 3.3 *Let (I, t), (I', t'), and (I'', t') be nodes such that I' and I'' have a common left endpoint, and $I' \supset I''$. If there are edges from (I, t) to both (I', t') and (I'', t'), then the edge from (I, t) to (I', t') can be deleted without affecting the length of the shortest path in the graph.*

Proof: Assume that τ is a path from source to sink containing the edge $((I, t), (I', t'))$ and assume $((I', t'), (I''', t'''))$ succeeds this edge in τ—if no such path exists, the edge is redundant and can be deleted harmlessly. Because $I' \supset I''$, there must also be an edge $((I'', t'), (I''', t'''))$ in the graph. Hence, we can replace the edges $((I, t), (I', t'))$ and $((I', t'), (I''', t'''))$ in τ by $((I, t), (I'', t'))$ and $((I'', t'), (I''', t'''))$ without affecting the length of τ. □

The repeated application of Lemma 3.3 to the graph (until no more edges can be deleted) leads to a reduced graph in which every node has just one outgoing edge per set of nodes with intervals with a common left endpoint and with a common fence type. The single edge from the initial graph that remains after the repeated application of Lemma 3.3 is the one to the node corresponding to the shortest interval. Since there are $O(|\Sigma|) = O(n)$ possible left endpoints and just two fence types, the number of outgoing edges from one node is reduced to $O(n)$. The total number of edges of the reduced graph is therefore $O(n^3)$.

Lemma 3.4 *The reduced graph contains $O(n^2)$ nodes and $O(n^3)$ edges.*

We can use Lemma 3.4 to compute fence designs efficiently. The approach is much more general though. It can be used for any part feeding problem for a part with $O(n)$ possible poses.

Theorem 3.5 *Let P be a part, let Σ be a set of possible (equilibrium) orientations of P. Let ϖ be a monotonic transfer function which is a mapping from Σ to Σ for a specific basic action of a part feeder. There exists a graph with $O(|\Sigma|^2)$ nodes and $O(|\Sigma|^3)$ edges such that a shortest path in the graph corresponds to a shortest feeder design which orients P up to symmetry if such a design exists. The nodes of the graph correspond to intervals of possible orientations of P, the edges of the graph connect nodes if there exists a basic feeder action such that the interval of the former can be mapped onto the latter.*

The computation of the (reduced) graph for fence design is rather simple. In the reduce graph, each node with interval $[\sigma_i, \sigma_j]$, has just one outgoing edge to the set of nodes with intervals with a common left endpoint $\sigma_{i'}$, and a common fence type. This outgoing edge connects the former node to the node with the shortest interval which can be reached by a valid reorientation of the push direction.
The shortest interval with left endpoint $\sigma_{i'}$ is derived by a push direction which maps σ_i onto $\sigma_{i'} - |l(\sigma_{i'})|$. The construction of the graph follows directly from this observation. We align the interval with the left environment of the reachable orientations for a valid reorientation of the push direction, and compute the resulting interval after application of the push function. If it is not possible to align σ_i with $\sigma_{i'} - |l(\sigma_{i'})|$, then we take the reorientation of the jaw such that we get as close as possible to $\sigma_{i'} - |l(\sigma_{i'})|$.
Consecutively computing the outgoing edges for one node can be carried out in linear time, by shifting $[\sigma_i, \sigma_j]$ along the possible reorientations of the push direction. Hence, the total time required to compute the graph edges is $O(n^3)$. A breadth-first search on the graph takes $O(n^3)$ time (see e.g. [37]). The number of nodes of the graph bounds the maximum length of a fence design. We yield the following theorem.

Theorem 3.6 *Let P be a polygonal part with n vertices. A fence design using the minimal number of fences which orients P up to symmetry can be computed in $O(n^3)$ time. The resulting design consists of at most $O(n^2)$ fences.*

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We simply pick the midpoint of every such interval as the reorientation, and get a push plan which is a fence design. We can easily compute the fence angles from the reorientation angles on the path, using the geometric model of the fence design.

3.3 An output-sensitive algorithm

A disadvantage of the algorithm in the previous section is the (high) running time of $O(n^3)$, even if the computed fence design turns out to be short. In the next section, we shall show that, for a large class of parts, the fence designs have linear length, and for most long and thin parts, the fence design has constant length. This suggests that an output-sensitive algorithm, whose running time depends on the number of fences in the design, is to be preferred. In this section we present an algorithm which calculates the shortest fence design in $O(kn \log n)$ time, with n the number of vertices of the part and k the number of fences of the computed design. Using this algorithm, most fence designs can be computed in $O(n^2 \log n)$ time, and even in $O(n \log n)$ time for most long and thin parts.

3.3.1 Maintaining the shortest intervals

The main idea of the algorithm is to maintain the shortest interval of possible orientations after k fences, instead of precomputing the whole graph of all possible intervals of orientations. This is basically the same technique as used by Goldberg's algorithm to compute push plans [45]. Goldberg maintains the interval of possible orientations, and greedily shrinks this interval per application of the pushing jaw. We, however, must take into account the constraints of fence design. It is not sufficient to maintain a single shortest interval of possible orientations. Lemma 3.3 indicates that it is sufficient to maintain for each stable orientation the shortest interval starting at this orientation. Also, to be able to compute the possible reorientation of the jaw, we have to have two copies, one for the shortest interval with a plan that ends with a left fence, and another copy for a plan ending with a right fence. We denote the right endpoint of the shortest interval starting with σ_i on a fence of type t after k fences by $\varsigma_i^t(k)$—the possible orientations after k fences, with the last fence of type t are contained in any interval $[\sigma_i, \varsigma_i^t(k)]$ for $i \in \{1, \ldots, |\Sigma|\}$. Let us denote the sets containing all intervals of possible orientations after k fences ending with a left or a right fence by $\Sigma^{\text{left}}(k)$ and $\Sigma^{\text{right}}(k)$ respectively.

After the first fence, the part can be in any orientation. Hence, we have the following sets of shortest intervals:

$$\begin{aligned}
\Sigma^{\text{left}}(1) &= \{[\sigma_1, \varsigma_1^{\text{left}}(1)], [\sigma_2, \varsigma_2^{\text{left}}(1)], \ldots, [\sigma_{|\Sigma|}, \varsigma_{|\Sigma|}^{\text{left}}(1)]\} \\
&= \{[\sigma_1, \sigma_{|\Sigma|}], [\sigma_2, \sigma_1], \ldots, [\sigma_{|\Sigma|}, \sigma_{|\Sigma|-1}]\} \\
\Sigma^{\text{right}}(1) &= \{[\sigma_1, \varsigma_1^{\text{right}}(1)], [\sigma_2, \varsigma_2^{\text{right}}(1)], \ldots, [\sigma_{|\Sigma|}, \varsigma_{|\Sigma|}^{\text{right}}(1)]\} \\
&= \{[\sigma_1, \sigma_{|\Sigma|}], [\sigma_2, \sigma_1], \ldots, [\sigma_{|\Sigma|}, \sigma_{|\Sigma|-1}]\}
\end{aligned}$$

We explain how to compute the sets of intervals $\Sigma^{\text{left}}(k)$ from $\Sigma^{\text{left}}(k-1)$ and $\Sigma^{\text{right}}(k-1)$. Computing the intervals in $\Sigma^{\text{right}}(k)$ is analogous. Let us consider the interval starting with orientation σ_i and suppose that we want to compute the orientation $\varsigma_i^{\text{left}}(k)$. Since the last fence that has been used is a left fence, we are allowed to change the push direction by angles in $(0, \pi/2)$ or $(\pi/2, \pi)$, if the $(k-1)$-th fence was a left or right fence respectively. This constrains the possible alignment of the previous calculated shortest intervals in $\Sigma^{\text{left}}(k-1)$ and $\Sigma^{\text{right}}(k-1)$ with σ_i. The preimage of σ_i under the push function is $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$. Only intervals which can be mapped onto this preimage can possibly result in an interval starting with σ_i and therefore are *candidates* for the shortest interval $[\sigma_i, \varsigma_i^{\text{left}}(k)] \in \Sigma^{\text{left}}(k)$. We have to determine the best interval among the candidate intervals from $\Sigma^{\text{left}}(k-1)$ and $\Sigma^{\text{right}}(k-1)$, i.e., the interval with the shortest possible image, which has to start with σ_i, under the shifted push function.

The image of a single candidate interval starting with σ_i can vary in length, depending on the position of the interval's left endpoint before applying the (shifted) push function. Any position of this left endpoint in $l(\sigma_i) \cup \{\sigma_i\} \cup r(\sigma_i)$ results in an image starting with σ_i. The image is shortest, though, when the candidate interval is aligned as close as possible to $\sigma_i - |l(\sigma_i)|$. Therefore, our first goal is to determine the intervals which can align with $\sigma_i - |l(\sigma_i)|$. We determine which intervals can be mapped onto angles in $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$. These two sets of intervals constitute the aforementioned candidates for the shortest interval $[\sigma_i, \varsigma_i^{\text{left}}(k)] \in \Sigma^{\text{left}}(k)$. The candidate of which the right endpoint, after proper alignment with σ_i, is leftmost determines the shortest interval $[\sigma_i, \varsigma_i^{\text{left}}(k)] \in \Sigma^{\text{left}}(k)$, and is therefore the best candidate. The position of the right endpoint of a candidate interval is dependent on the length of the interval, as well as on the leftmost position onto which we can possibly map the left endpoint of this interval with a feasible change of the push direction. Figure 3.4 depicts the orientation σ_i together with five candidate intervals, of which two cannot be mapped onto $\sigma_i - |l(\sigma_i)|$. The alignment of the intervals closest to

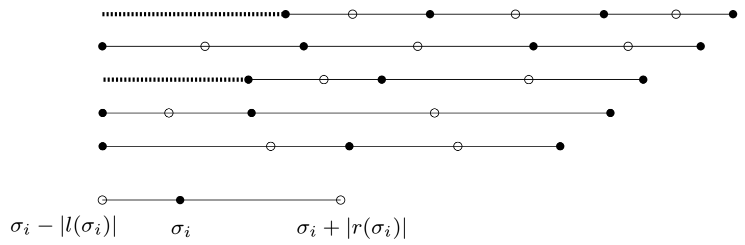


Figure 3.4 The candidate intervals for $[\sigma_i, \varsigma_i^{\text{left}}(k)]$. The two intervals with the trailing dashed line cannot be mapped onto $\sigma_i - |l(\sigma_i)|$ by a valid fence design. The best interval is the interval with the leftmost right endpoint in the figure.

$\sigma_i - |l(\sigma_i)|$ is depicted in the picture. The two dashed lines correspond to the difference between $\sigma_i - |l(\sigma_i)|$ and the best alignment of the intervals.
A naive, but simple, algorithm to compute the sets $\Sigma^{\text{left}}(k)$ and $\Sigma^{\text{right}}(k)$ is:

1. Test for each pair of an orientation σ_i and a fence type, all intervals in $\Sigma^{\text{left}}(k-1)$ and $\Sigma^{\text{right}}(k-1)$ and check if there is a valid reorientation of the jaw such that these intervals align with $\sigma_i - |l(\sigma_i)|$, or possibly map onto $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$ otherwise.
2. Determine the interval with the smallest possible right end point with respect to its best alignment with $\sigma_i - |l(\sigma_i)|$.
3. Compute the image of the push function of this interval and store this image in appropriate set $\Sigma^{\text{left}}(k)$ or $\Sigma^{\text{right}}(k)$.

This way, it takes linear time to determine the best best candidate per interval. Therefore, the naive approach leads to an $O(kn^2)$ algorithm. In the next section we will improve the time required to compute the sets $\Sigma^{\text{left}}(k)$ and $\Sigma^{\text{right}}(k)$.

3.3.2 Updating in $O(n \log n)$ time

We now show how to speed up the part of the algorithm which determines the best candidate from linear to logarithmic time per interval. To achieve this, we maintain a *range tree* (see e.g. [10]) storing the shortest intervals. Assume that we have a range trees, $\mathcal{T}^{\text{left}}(k-1)$, and $\mathcal{T}^{\text{right}}(k-1)$ on the left endpoints of the intervals in $\Sigma^{\text{left}}(k-1)$ and $\Sigma^{\text{right}}(k-1)$ respectively. A range tree is a balanced binary search tree which allows us to find the candidate intervals for any starting orientation after k fences in logarithmic time. The leaves of the tree contain the intervals of the previous step, ordered on starting orientation. The

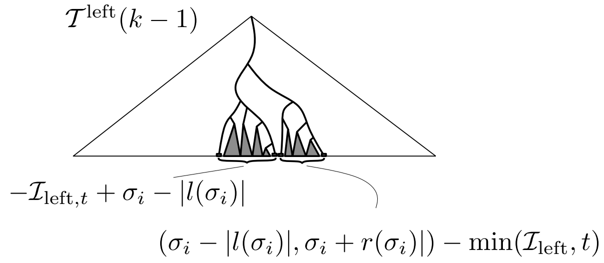

$\mathcal{T}^{\text{left}}(k-1)$
$-\mathcal{I}_{\text{left},t} + \sigma_i - |l(\sigma_i)|$
$(\sigma_i - |l(\sigma_i)|, \sigma_i + r(\sigma_i)|) - \min(\mathcal{I}_{\text{left}}, t)$

Figure 3.5 The two queries from the range tree $\mathcal{T}^{\text{left}}(k-1)$ for $\varsigma_i^t(k)$. The intervals from the query $-\mathcal{I}_{\text{left},t} + \sigma_i - |l(\sigma_i)|$ can be aligned with $\sigma_i - |l(\sigma_i)|$. The intervals from the query $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|) - \min(\mathcal{I}_{\text{left},t})$ correspond to intervals that receive a penalty.

nodes of the tree contain splitting values, which guide the search. The subset of keys which are contained in the subtree is referred to as the *canonical subset* of this subtree. Let us store in each node of the range tree, the position of the leftmost right endpoint of the canonical subset of intervals of the subtree rooted at this node. Furthermore, we store the length of the shortest interval of the same canonical subset. We can now easily derive the candidate interval with the leftmost right-interval, thereby reducing the number of candidates to $O(\log n)$, as we shall outline in the following paragraphs.

Let us first try to understand which range queries are necessary and sufficient to compute a new interval. Assume that we want to compute $\varsigma_i^t(k)$, i.e., the right endpoint of the shortest interval starting with σ_i after k fences; the k'th fence being of type t. We first analyze which intervals from $\Sigma^{\text{left}}(k-1)$ align with $\sigma_i - |l(\sigma_i)|$, and which intervals map onto $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$. The interval from $\Sigma^{\text{left}}(k-1)$ must be reoriented by an angle in $\mathcal{I}_{\text{left},t}$. Intervals $[\sigma_j, \sigma_{j'}]$ with $\sigma_j \in \sigma_i - |l(\sigma_i)|$ minus angles in $\mathcal{I}_{\text{left},t}$ can be mapped onto $\sigma_i - |l(\sigma_i)|$. If, on the other hand, the minimal angle $\min(\mathcal{I}_{\text{left},t})$ in $\mathcal{I}_{\text{left},t}$, cannot map σ_j onto $\sigma_i - |l(\sigma_i)|$, σ_j can be mapped onto $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$, if $\sigma_j + \min(\mathcal{I}_{\text{left},t}) \in (\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$. We need similar intervals from $\Sigma^{\text{right}}(k-1)$. Therefore, we query as follows from $\mathcal{T}^{\text{left}}(k-1)$ and $\mathcal{T}^{\text{right}}(k-1)$: The intervals which align with $\sigma_i - |l(\sigma_i)|$ correspond to two range queries of length $\pi/2$; $-\mathcal{I}_{\text{left},t} + \sigma_i - |l(\sigma_i)|$ in $\Sigma^{\text{left}}(k-1)$, and $-\mathcal{I}_{\text{right},t} + \sigma_i - |l(\sigma_i)|$ in $\Sigma^{\text{right}}(k-1)$. Since these intervals align with $\sigma_i - |l(\sigma_i)|$, the shortest such interval (which is stored with the nodes) is the candidate per canonical subset. There are a logarithmic number of canonical subsets, and consequently a logarithmic number of candidates.

The intervals which can only be mapped onto $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$ receive a penalty which is added to the length of this interval. In the end, the interval which has the leftmost right endpoint is the best candidate of a canonical subset.

There is a minor problem with calculating the leftmost right endpoint of a cyclic set of intervals. The penalty of two consecutive intervals might differ 2π plus the desired difference. To overcome problems with circularity, compute two copies of the push function and store the both of them in the range tree—in the second copy of the push function the values are 2π higher than in the first copy. This way, any range of intervals can be found as one piece of the range tree. We query $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|) - \min(\mathcal{I}_{\text{right},t})$ in $\mathcal{T}^{\text{right}}(k-1)$ and $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|) - \min(\mathcal{I}_{\text{left},t})$ in $\mathcal{T}^{\text{left}}(k-1)$. To overcome problems with circularity, we make sure that $\sigma_i + |r(\sigma_i)| > \sigma_i - |l(\sigma_i)| > \min(\mathcal{I}_{\text{right},t})$, by adding 2π to some of the values, if necessary. We derive a logarithmic number of canonical subsets. The values of their leftmost right endpoints are too large to compete with the lengths of the candidates of the former two queries. We want the difference between $\sigma_i - |l(\sigma_i)|$, and the right endpoint. Therefore, we subtract $\sigma_i - |l(\sigma_i)|$ from the found right endpoints, and derive a logarithmic number of appropriate candidates.

In Figure 3.5 the queries from $\mathcal{T}^{\text{left}}(k-1)$ for $\varsigma_i^t(k)$ are depicted. The query on the left corresponds to intervals of which the starting orientation can be mapped onto $\sigma_i - |l(\sigma_i)|$. The length of the shortest interval of the canonical subsets of this query determines the best candidate for this query. The query on the right corresponds to intervals of which the starting orientation can be mapped onto $(\sigma_i - |l(\sigma_i)|, \sigma_i + |r(\sigma_i)|)$. The difference of the leftmost right endpoint of the canonical subsets of this query and $\sigma_i - |l(\sigma_i)|$ determines the best candidate for this query. Together with the corresponding queries from $\mathcal{T}^{\text{right}}(k-1)$, the shortest interval starting with σ_i on a fence of type t is determined. The total number of candidates resulting from the four range queries is $O(\log n)$. The best candidate of the four queries is the interval which determines the value of $\varsigma_i^t(k)$, and is computed in $O(\log n)$ time. This leads to the following lemma.

Lemma 3.7 *Let there be two range trees; one in which two copies of two folds of the unit circle of the intervals in the set* $\Sigma^{\text{left}}(k-1)$ *are stored, and one for the intervals in the set* $\Sigma^{\text{right}}(k-1)$*, stored in a similar way. Both trees are ordered on the left endpoints of the intervals. If in the nodes the shortest interval as well as the leftmost right endpoint of the corresponding canonical subsets are stored, then the sets* $\Sigma^{\text{left}}(k)$ *and* $\Sigma^{\text{right}}(k)$ *can be computed in* $O(n \log n)$ *time.*

The construction of the range tree takes $O(n \log n)$ time. We presort the intervals on their length, and their right endpoint. Then, we build the range-tree bottom up in linear time. The query takes $O(\log n)$ per calculated new shortest interval. We have $O(n)$ new intervals, and per fence we spend $O(n \log n)$ time.

3.3.3 Incremental computation of the fence design

In this section, we put together the pieces of the previous sections, and show how to incrementally compute a fence design. In the previous section, we showed how to compute intervals of possible orientations while incrementing the number of fences in a design.

Obviously, we want to know when to stop extending the design length, i.e., our algorithm has to terminate when the part is oriented up to symmetry. If the part has no (periodic) symmetry, then the part is oriented if there is one stable orientation in an interval of possible orientations. If, on the other hand the part has symmetry, then we have to detect if the part is oriented up to symmetry. This can be done in constant time as well, by comparing the indices of the computed intervals. We can precompute the desired difference of the indices with the aid of a string matching algorithm which tries to find the shifted push function in the concatenation of two copies of the push function. In the second part of this chapter we shall show that there always exists a fence design that orients a part up to symmetry—see Theorem 3.10. Using this result here, we conclude that our algorithm will always find such a design, and thus terminate.

We want to know which fence design corresponds to our final interval. This can be accomplished by incrementally building the same graph as in Section 3.2, storing the used fences with the edges of the graph. We can find the fence design, by tracking back the path from the 'oriented' node to the source of the graph. The size of the graph increases by $O(n)$ per step of the algorithm. Tracking back the path takes $O(k)$ time. The following theorem summarizes the result of this section.

Theorem 3.8 *Let P be a polygonal part with n vertices. A fence design of minimal length which orients P up to symmetry can be computed in $O(kn \log n)$ time, where $k = O(n^2)$ is the number of fences in the resulting design.*

Proof: From Theorem 3.10 it follows that a fence design which orients P up to symmetry exists. We can, by subtracting the indices of a current shortest interval, check in constant time if the part is oriented up to symmetry. Extending the fence design by one fence takes $O(n \log n)$ time. Maintaining the used fences takes $O(n)$ time per extension. The total running time is, therefore, $O(kn \log n)$ time, with k the number of fences in the design. □

Clearly, the running time of the incremental algorithm depends on the length of the final design. The best known upper bound on the length of a fence design follows from the size of the graph of the non-incremental algorithm and is $O(n^2)$. Hence, the incremental algorithm might be out-performed by the graph based algorithm. In the next section we shall see that most fence designs have linear lenght. Hence, for most parts, the running time of the incremental algorithm is $O(n^2 \log n)$, which is better than the asymptotic time bound on the running time of the graph based algorithm. We conjecture that for any polygonal part, there exists a fence design of linear length.

3.4 Completeness for asymmetric parts

In the first part of this chapter, we have seen how to compute fence designs for a given part. We will now show that there always exists a fence design which orients a given part up to symmetry. In this section we concentrate on parts with push functions with a unique longest left environment $l(\sigma_i)$ and a unique longest right environment $r(\sigma_j)$. We prove that for these *asymmetric* parts a fence design always exists and has length $O(n)$. In Section 3.5 we deal with arbitrary parts.

We use the push plans of Chen and Ierardi [32] that we presented in Section 2.2.1, as a basis for the plans in this section. We recall from Section 2.2.2 that we denote the length of the longest right environment by λ, and the length of the longest left environment by $\reflectbox{$\lambda$}$. Chen and Ierardi have shown that a total of $|\Sigma| + 1$ basic actions suffice to put P into orientation σ^* for which $|r(\sigma^*)| = \lambda$—the sequence $(\lambda - \epsilon)^{|\Sigma|+1}$ is a valid push plan for P. In order for the push plan $(\lambda - \epsilon)^{|\Sigma|+1}$ or $(-\reflectbox{$\lambda$} + \epsilon)^{|\Sigma|+1}$ to be a valid fence design, we have to show that it satisfies the constraints formulated in Definition 3.1. We observe first of all that there can be no more than three environments of length at least $\pi/2$, because the longest two left environments have different lengths and the longest two right environments have different lengths. As a result, there is at most one left environment of size at least $\pi/2$ or at most one right environment of size at least $\pi/2$. Assume without loss of generality that there is at most one right environment of size at least $\pi/2$. Although the length λ of the longest right environment $r(\sigma)$ can be at least $\pi/2$, the length λ' of the second largest right environment $r(\sigma')$ must be smaller than $\pi/2$. If we now choose ϵ such that $\lambda' < \lambda - \epsilon < \min\{\lambda, \pi/2\}$, then we get that $\lambda - \epsilon > |r(\sigma_i)|$ for $i \in [1, \ldots, |\Sigma|]$. In addition, it clearly holds that $\lambda - \epsilon < \pi/2$, which makes it easy to verify that $(\lambda - \epsilon)^{|\Sigma|+1}$ is a fence design.

Theorem 3.9 *An asymmetric polygonal part* P *with* n *vertices can be oriented by a fence design of length* $O(n)$*.*

Van der Stappen *et al.* [100] showed that most long and thin parts can be oriented by a fence design consisting of $O(1)$ fences.

3.5 Arbitrary parts

The considerations in Section 3.4 show that we can orient a part by a linear-length fence design if its push function has a unique longest left or right environment for which the second largest interval has a length smaller than $\pi/2$. For asymmetric parts, there always exists such an environment. If we deal with arbitrary parts, there can be several environments with the same size λ, even with size greater than $\pi/2$. In this section we show that for every polygonal part there is a fence design that orients the part up to symmetry in its push function. We recall that $|\Sigma|$ denotes the number of stable (equilibrium) orientations of P. We show that we can orient P if the period of the push function is 2π. The plans can actually be used to orient any part up to the period in its push function. The method we use is similar to the method Chen and Ierardi introduced to generate push plans [32]. Recall that a fence design is a push plan satisfying constraints on the reorientations of the jaw. We will try to produce push plans that only use reorientations in either $(0, \pi/2)$ or $(-\pi/2, 0)$, as such plans clearly satisfy Definition 3.1. There are two problems with the implementation of the push plans of Chen and Ierardi as a fence design. First, a problem occurs if there are multiple right environments with size greater than $\pi/2$: the simple plans of Chen and Ierardi that use reorientations of the jaw of $\lambda - \epsilon$ do not satisfy the constraints of fence design. Second, there is a problem if there is no unique largest right environment. In this case, Chen and Ierardi apply their 'Stretching Lemma' which in general uses any reorientation of the pushing jaw. The Stretching Lemma shows that we can shift two possible orientations which both have a maximal left or right environment closer to each other with one single push. In other words, we can break the symmetry if there are multiple orientations with equally long maximal left or right environments. For a push plan Lemma 2.7 remains valid [32, 100]. For fence design this is not necessarily the case.

We can reduce the possible orientations of the part to those with maximal right environments or with right environments of length greater than $\pi/2$ by an alternating sequence of jaw applications and jaw reorientations by an angle $\min\{\lambda - \epsilon, \pi/2 - \epsilon\}$ (λ and ϵ as in Section 3.4; no right environment shorter than $\pi/2$ is longer than $\pi/2 - \epsilon$). (We can similarly reduce the possible orientations to orientations with such left environments.) As observed, for arbitrary parts, the number of possible orientations can be larger than one. We have to use more sophisticated fence designs to further reduce the number of possible orientations. We start with some definitions which we use throughout the rest of this section. Let λ again be the length of the longest right environment and let $\reflectbox{$\lambda$}$ be the length of the longest left environment. Furthermore, let $\Sigma = \sigma_1 \ldots \sigma_{|\Sigma|}$ be the cyclic sequence of stable equilibria, cut at some arbitrary orientation and ordered by *increasing* angle. The left cycle of the sequence Σ of stable equilibria is the smallest c_l for which $|l(\sigma_i)| = |l(\sigma_{i+c_l})|$ for all $0 \leq i < |\Sigma|$. Similarly, the right cycle of the sequence Σ of stable equilibria is the smallest c_r for which $|r(\sigma_i)| = |r(\sigma_{i+c_r})|$. Indexing is modulo $|\Sigma|$. If the cycle of set of environments of equals $|\Sigma|$, we call the set *acyclic*. The set is called cyclic, otherwise. We denote by Σ^{λ} the ordered set of orientations in Σ with right environments of length λ, or length greater than or equal to $\pi/2$. We define the right measure $M_{\Sigma^{\lambda}}$ of an interval $[\sigma_i, \sigma_j]$ of orientations by $M_{\Sigma^{\lambda}}([\sigma_i, \sigma_j]) = |\{\sigma \in \Sigma^{\lambda} | \sigma_i \leq \sigma < \sigma_j\}|$. In a similar manner, we can define a set $\Sigma^{\reflectbox{$\lambda$}}$ of orientations with left environments of length $\reflectbox{$\lambda$}$, and a left measure $M_{\Sigma^{\reflectbox{$\lambda$}}}$. We let $\epsilon > 0$ be a small constant such that $\lambda - \epsilon$ and $\lambda + \epsilon$ are both smaller than $\pi/2$ but larger than any environment of length less than $\pi/2$. In addition, the constant ϵ is smaller than the length of any left or right environment. We bear in mind that polygonal parts without meta-stable edges have push functions without left and right environments of zero length.

In the following, we shall show how to shorten the length of an interval of possible orientations, until this interval contains only one stable orientation of P. After the first fence in the design, P can be in any of the orientations in Σ. In other words, the part is in any of the orientations of an interval $[\sigma_i, \sigma_{i-1}]$, which contains all orientations in Σ. Our goal is now to reduce the length of the interval of possible orientations. Slightly abusing the 'size of the output notation', we denote the interval of possible orientations after k fences by $[\varsigma_F(k), \varsigma_L(k)]$. Hence, $[\varsigma_F(1), \varsigma_L(1)] = [\sigma_i, \sigma_{i-1}]$, for some $\sigma_i \in \Sigma$.

Our push plans for arbitrary parts consist of three types of basic building blocks. These building blocks are referred to as MOVE, SHIFT, and REDUCE.

- Suppose that $[\varsigma_F(k), \varsigma_L(k)]$ is the current interval of possible orientations. Then, the push plan $(\min(\lambda, \pi/2) - \epsilon)^{|\Sigma|}$ MOVEs the interval $[\varsigma_F(k), \varsigma_L(k)]$ into an interval $[\varsigma_F(k + |\Sigma|), \varsigma_L(k + |\Sigma|)]$ with $\varsigma_F(k + |\Sigma|), \varsigma_L(k + |\Sigma|) \in \Sigma^{\lambda}$ of equal right measure.
- Suppose that $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$, is the current interval of possible orientations. Then, the push plan $(|r(\varsigma_F(k))| + \epsilon)$ SHIFTs the interval $[\varsigma_F(k), \varsigma_L(k)]$ into the interval $[\varsigma_F(k+1) = \sigma_{i+1}, \varsigma_L(k+1)]$. If $\sigma_i, \sigma_j \in \Sigma^{\lambda}$ then $\varsigma_L(k+1) = \sigma_{j+1}$, and the right measure of $[\varsigma_F(k+1), \varsigma_L(k+1)]$ equals the right measure of $[\varsigma_F(k), \varsigma_L(k)]$. Notice that in that case a SHIFT followed by

a MOVE maps $[\varsigma_F(k), \varsigma_L(k)]$ into an interval $[\varsigma_F(k+|\Sigma|+1), \varsigma_L(k+|\Sigma|+1)]$ with $\varsigma_F(k+|\Sigma|+1), \varsigma_L(k+|\Sigma|+1) \in \Sigma^\lambda$ and $\varsigma_F(k+|\Sigma|+1) \neq \sigma_i$, $\varsigma_L(k+|\Sigma|+1) \neq \sigma_j$ (provided that Σ^λ has at least two elements) of equal right measure.

- Suppose that $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$ with $\sigma_i, \sigma_j \in \Sigma^\lambda$ is the current interval of possible orientations. We want to define an operation which REDUCEs this interval to some interval $[\varsigma_F(k+|\text{REDUCE}|), \varsigma_L(k+|\text{REDUCE}|)]$ of smaller measure. The REDUCE exploits specific asymmetries that are present in the push function to achieve the reduction of the measure. Different classes of push functions lead to different REDUCEs.

The objective is to use these building blocks in a push plan that is guaranteed to result in an interval of possible orientations of measure zero. From Lemma 2.6, it follows that if $\varsigma_F(k), \varsigma_L(k) \in \Sigma^\lambda$, then there exists a push plan such that $M_{\Sigma^\lambda}([\varsigma_F(k+2), \sigma_j(k+2)]) < M_{\Sigma^\lambda}([\varsigma_F(k), \varsigma_L(k)])$. The proof of the lemma uses the monotonicity of the push functions and the definition of the shifted push function. The problem in applying these ideas in fence design is that the two required reorientations of the push direction may not be achievable by a sequence of fences. We will use combinations of the SHIFT and MOVE plans to overcome this problem.

We classify the push functions, based on the left and right cycle of the stable equilibria, and the sizes of the environments. The implementation of the REDUCE is the main difference between the distinct classes of push functions. We distinguish the following classes of push functions that do not correspond to the asymmetric parts of Section 3.4. The remaining class of parts are subject to Theorem 3.9.

1. (a) The left environments are acyclic and no left environment is longer than $\pi/2$.
   (b) The right environments are acyclic and no right environment is longer than $\pi/2$.
2. Both the left and the right cycle are cylic.
3. Both the left and the right cycle are acyclic, and there is more than one environment of length greater than $\pi/2$.

Let us first concentrate on push functions in the first and second class. Suppose that $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$, $i, j \in \{1, \ldots, |\Sigma|\}$, $\sigma_i, \sigma_j \in \Sigma^\lambda$, is the current interval of possible orientations. Let $\sigma_{\tilde{\imath}}$ and $\sigma_{\tilde{\jmath}}$ be the next orientations in Σ^λ, succeeding σ_i and σ_j respectively. If the sequence of right environments of orientations between σ_i and $\sigma_{\tilde{\imath}}$ differs from the sequence of right environments between σ_j and $\sigma_{\tilde{\jmath}}$, then there exists a simple strategy that maps $[\sigma_i, \sigma_j]$ either onto $[\sigma_{\tilde{\imath}}, \sigma_{\bar{\jmath}}]$ with $\sigma_{\bar{\jmath}} \in \Sigma \setminus \Sigma^\lambda$ and $\sigma_j < \sigma_{\bar{\jmath}} < \sigma_{\tilde{\jmath}}$ or onto $[\sigma_{\bar{\imath}}, \sigma_{\tilde{\jmath}}]$ with $\sigma_{\bar{\imath}} \in \Sigma \setminus \Sigma^\lambda$ and $\sigma_i < \sigma_{\bar{\imath}} < \sigma_{\tilde{\imath}}$. Moreover, it can be shown that any interval $[\sigma_i, \sigma_j]$ with $\sigma_i, \sigma_j \in \Sigma^\lambda$ can be transformed—by means of SHIFT and MOVE plans—into a different interval $[\sigma_{i'}, \sigma_{j'}]$ with $\sigma_{i'}, \sigma_{j'} \in \Sigma^\lambda$ of equal measure, such that the simple strategy is guaranteed to map $[\sigma_{i'}, \sigma_{j'}]$ onto $[\sigma_{\tilde{\imath}'}, \sigma_{\bar{\jmath}'}]$, with $\sigma_{\bar{\jmath}'} \in \Sigma \setminus \Sigma^\lambda$ and $\sigma_{j'} < \sigma_{\bar{\jmath}'} < \sigma_{\tilde{\jmath}'}$, with $\sigma_{\tilde{\imath}'}$ and $\sigma_{\tilde{\jmath}'}$ in Σ^λ succeeding $\sigma_{i'}$ and $\sigma_{j'}$ respectively. A subsequent reorientation by $\lambda + \epsilon$ followed by a push will eliminate $\sigma_{\tilde{\imath}'}$ as a possible orientation of the part without adding new possible orientations from Σ^λ to the interval of possible orientations. As a consequence, the measure of the resulting interval of possible orientations is smaller than $M_{\Sigma^\lambda}([\sigma_{\tilde{\imath}'}, \sigma_{\tilde{\jmath}'}]) = M_{\Sigma^\lambda}([\sigma_i, \sigma_j]$. For push functions from the first class, this strategy (or its equivalent using left environments) suffices to reduce the interval of possible orientations to an interval of measure zero. The reorientations of the push direction in the entire scheme are restricted to $(0, \pi/2)$ or $(-\pi/2, 0)$, which makes the sequence of pushes a valid fence design. If the left and right cycle are both smaller than $|\Sigma|$, then we must eventually switch our attention from the right environments to left environments to break the symmetry of the right environments (or vice versa). It turns out that such a switch can be accomplished without violating the reorientation constraints for fence designs.

The third class of push functions requires some modifications to the MOVE, SHIFT, and REDUCE framework. There are, however, at most three environments of length greater than $\pi/2$, which makes it possible to treat the different cases one by one, and provide dedicated push plans which use both left and right fences. These dedicated push plans satisfy the reorientation constraints and are therefore valid fence designs. The three classes of push functions will be dealt with in Appendix A. We there show the details of the MOVE, SHIFT and REDUCE framework for the three different classes of parts that are distinguished. Theorem 3.10 summarizes the result of this section.

Theorem 3.10 *Any polygonal part can be oriented up to symmetry by a fence design.*

A consequence of Theorem 3.10 is that the algorithm for computing fence designs presented in Section 3.2 always outputs a fence design. Since the graph used in the algorithm has $O(n^2)$ nodes, the length of the shortest path is $O(n^2)$.

Corollary 3.11 *Any polygonal part with* n *vertices can be oriented up to symmetry by a fence design of length* $O(n^2)$. *The optimal fence design can be computed in* $O(n^3)$ *time.*

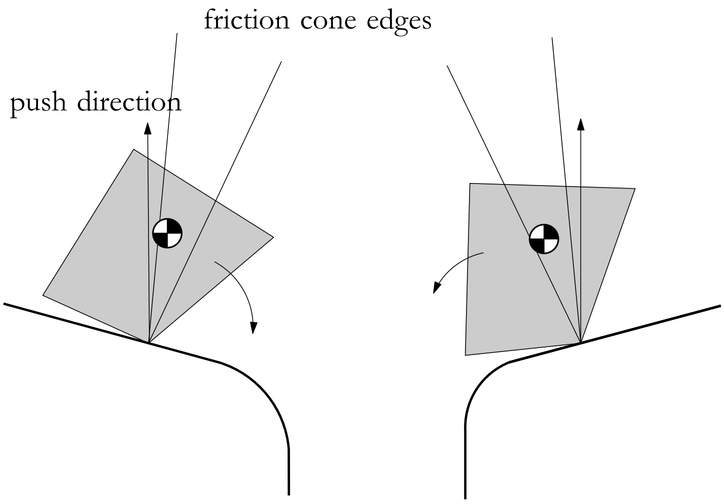


Figure 3.6 A part on a left and right fence with friction. Despite the similarity in orientation with respect to the fence, the left fence rotates the part to another edge than the right fence.

3.6 Friction

The remaining part of this chapter studies two extensions of fence design. In this section, we incorporate friction, and in the next section, we will focus on modular fences.

3.6.1 The model

In Section 2.1.1 the reader can find a discussion of the behavior of the part when pushed by a jaw in the plane. We shall incorporate the role of friction into fence design accordingly. The coefficient of friction is denoted by μ.
The first observation we make is that we cannot make fences which are very shallow, because due to friction these fences do no longer allow the parts to slide off. The (absolute) minimum fence angle must be strictly larger than $\arctan \mu$.
The second difference is that the part will align with the fence in a different way. Under the assumption that the fence angle is large enough, the line which separates area **II** from area **III** in Figure 2.2 is the left edge of the friction cone, if the part is on a left fence, and the right edge of the friction cone otherwise. This implies that the push function, which predicts the rotation of the part, is no longer symmetric. There are orientations for which a left fence rotates the part

clockwisely, and a right fence rotates the part counterclockwisely (see Figure 3.6). However, the push function for a part on a left fence is still monotonic, as is the push function for a part on a right fence. Moreover, one could look at the fence with friction as a frictionless fence which is $\arctan \mu$ shallower. The normal of this shallower frictionless fence coincides with described edge of the friction cone. However, the steepest frictionless fence which corresponds to a fence with friction is $\pi/2 - \arctan \mu$ for a right fence, and $\arctan \mu - \pi/2$ for a left fence. Unfortunately, this implies that fences with friction are not able to orient any part up to symmetry. It is possible to construct parts which can only be oriented with a reorientation of the push direction arbitrarily close to π [2]. Since the reorientation of the jaw closest to π is $\pi - \arctan \mu$ for a fence with friction, we can always construct a part which cannot be oriented by fences with friction.

3.6.2 Algorithms for fence designs with friction

In this section we discuss two algorithms that compute a fence design with friction, if such a design exists. Firstly, we discuss a modification of the graph based algorithm. Secondly we give a modification of the output sensitive algorithm. Let us first redefine the intervals of possible reorientations of the jaw for fences with friction. The next table gives the resulting possible reorientations of the jaw, after taking friction into account.

t_i	$\alpha_{i+1} \in \mathcal{I}^{\mu}_{t_i,t_{i+1}}$	t_{i+1}
left	$(0, \pi/2 - \arctan \mu)$	left
left	$(\pi/2, \pi - \arctan \mu)$	right
right	$(-\pi + \arctan \mu, -\pi/2)$	left
right	$(-\pi/2 + \arctan \mu, 0)$	right

A transition of the part from one fence to another gives us the possibility to implement a reorientation of the push direction that is contained in the corresponding angular interval from the table. This means that we can still use our graph-based algorithm. However, we have to modify the component which decides if a node can be connected to another node in the graph.

Consider two graph nodes (I, t) and (I', t'), where I and I' are intervals of stable equilibria and t and t' are fence types. Let $\mathcal{I}^{\mu}_{t,t'}$ be the open interval (of length $\pi/2 - \arctan \mu$) of reorientations admitted by the successive fences of types t and t' according to the table above. There is a directed edge from (I, t) to (I', t') if there is an angle $\alpha \in \mathcal{I}^{\mu}_{t,t'}$ such that a reorientation of the push direction by α followed by a push moves any stable orientation in I into a stable orientation in I', and furthermore α is constructed by a fence with friction. To check this condition, we determine the preimage $(\phi, \psi) \supseteq I'$ of I', under the push function. Observe that if $|I| = \sigma_j - \sigma_i < \psi - \phi$, any reorientation in the open interval $(\phi - \sigma_i, \psi - \sigma_j)$ followed by a push will map I into I'. We add an edge from (I, t) to (I', t') if the intersection of $(\phi - \sigma_i, \psi - \sigma_j)$ and the interval $\mathcal{I}^{\mu}_{t,t'}$ of admitted reorientations is not empty. We label this edge with this intersection. Also, we add a source and a sink to the graph. We connect the source to every node (I, t) in the graph for which I contains all $|\Sigma|$ stable equilibria, and we connect every node (I, t) with I an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to a shortest such path and can be found by a breadth-first search from the source.

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We pick the midpoint of every such interval, and determine if this reorientation is by a left or a right fence. For a left fence, we add $\arctan \mu$ to the corresponding fence angle. For a right fence, we subtract $\arctan \mu$. We obtain a fence design with friction.

Theorem 3.12 *Let P be a polygonal part, with n vertices. There is an algorithm which computes a shortest fence design with friction in $O(n^3)$ time, if there is a fence design with friction which orients P. If no fence design exists, the algorithm reports failure in the same time bound.*

We can also modify the output-sensitive algorithm to compute the shortest fence design for fences with friction, if a design exists. The modification by itself is quite straightforward. Instead of using the table of Section 3.1, we use the modified table of this section to determine the possible reorientations of the jaw to extend the fence designs. In the case of fences with friction, we do not know in advance whether a design exists. Therefore, we have to add an extra stop criterion to ensure that the algorithm does not try to extend the fence design when we discover no design exists.

Let us concentrate on the array which stores the current set of possible intervals. During one step of the algorithm, we replace the intervals that are stored in the of the array by shorter intervals. If an interval is not replaced in a step of the algorithm, then we need not to compute the new intervals that are derived from this interval in the succeeding step. This is because the length of the intervals monotonically decrease. Hence, we stop the execution of the algorithm when during a single step no interval is improved. It is not hard to see that we can check whether at least one interval is improved in a single step of the algorithm without increasing the asymptotic running time. Since there are at most $O(n^2)$ intervals of stable equilibria, the algorithm terminates after at most $O(n^2)$ steps. This leads to an upper bound on the running time of $O(n^3 \log n)$. While improving the intervals, we also incrementally construct the graph corresponding to the found fence designs. At the end, we compute the fence design with friction from a path in the graph in the same way as in the original graph based method.

Theorem 3.13 *Let P be a polygonal part with n vertices. If there exists a fence design with friction that orients P, then a design of minimal length can be computed in $O(kn \log n)$ time, with k the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm reports failure in $O(n^3 \log n)$ time.*

If we could prove that any fence design has linear length, then this bound would most likely have impact to the algorithm of this section, and result in an upper bound on the running time of $O(n^2 \log n)$ of the algorithm for computing fence designs with friction.

3.7 Modular fence designs

In this section, we discuss modular fence designs, i.e., designs for which the fence angles are to be taken from a discrete and finite set of angles. Modular fences can be useful in an industrial application of a conveyor to orient parts. There is no need for ‘expensive’ adjustable fences. The conveyor operator picks prefabricated fences out of a box and just attaches them along the sides of the belt. Figure 3.7 shows an example of a fence design constructed with modular fences of $\pi/4$ and $-\pi/4$, together with a fence design build with arbitrary fence angles. It is obvious that the modular design is at least as long as the design with arbitrarily oriented fences.

3.7.1 The model

We first discuss how the discretization of the possible fence angles influences the possible reorientations of the jaw, and therefore the algorithms to compute fence designs.

Througout this section, let m denote the number of fence angles. Every fence can correspond to two reorientations of the jaw. A left fence with fence angle

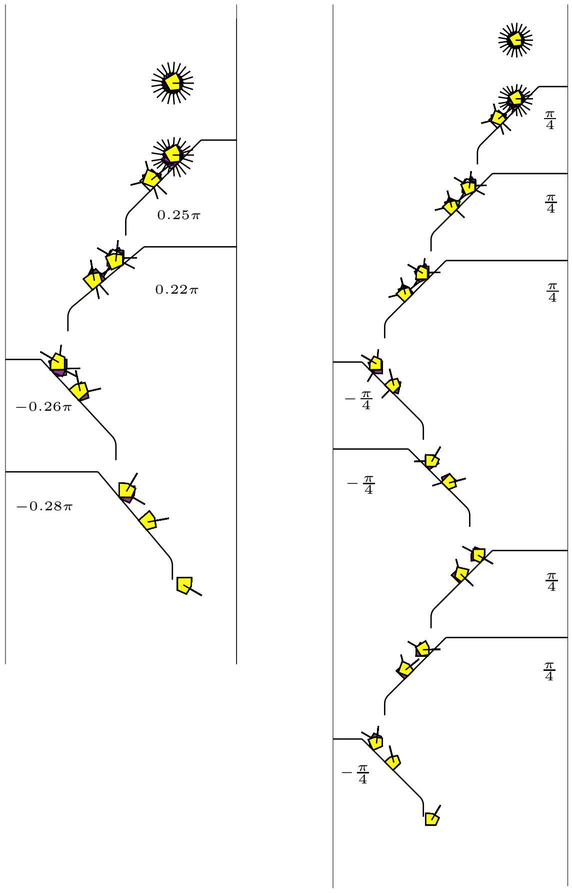


Figure 3.7 Two fence designs for the same part. On the left, any fence angle is available. On the right, only fences of $\pi/4$ and $-\pi/4$ are used. The line segments emanating from the center-of-mass of the parts represent possible orientations of the part

$\phi \in (-\pi/2, 0)$ corresponds to a reorientation of the jaw of $\phi + \pi/2$ if it succeeds a left fence in the fence design, and $\phi - \pi/2$ if it succeeds a right fence. A right fence with fence angle $\phi \in (0, \pi/2)$ corresponds to a reorientation of the jaw of $\phi + \pi/2$ if it succeeds a left fence in the fence design, and $\phi - \pi/2$ if it succeeds a right fence. The total number of possible reorientations of the jaw is bounded by twice the number of available fence angles, which is $2m$.

As indicated before, it is possible to construct parts which can only be oriented with a reorientation of the push direction push arbitrarily close to π [2]. This implies that one can always construct a part that cannot be oriented, given a set of modular fence angles.

The modular fences we treat in this section are frictionless. Using the line of reasoning of Section 3.6, one can easily modify the algorithms of this section to compute modular fence designs with friction as well.

3.7.2 Algorithms for modular fence designs

In this section we give two algorithms to compute modular fence designs (if they exist for the part). As in the section on fences with friction, the first algorithm is based on the graph based algorithm of Section 3.2, and the second is based on the output-sensitive algorithm of Section 3.3. However, the running times of the algorithms are surprisingly different. The former runs in $O(mn^2)$ and the latter in $O(mkn)$ time, where k the number of fences in the design. Since k is $O(n^2)$ in the worst case, but can be bounded by $O(n)$ for most parts, and even $O(1)$ for many long and thin parts, it is interesting to present both algorithms.

In the algorithms of this section, we want to be able to quickly compute the image under the push function of a value of the form $\sigma + \alpha$, where σ is a stable equilibrium, and α is taken from in the set of reorientations of the jaw by modular fences. Therefore, we precompute these values using the push function. The number of possible reorientations of the jaw is $2m$. These images are computed by tracing the push function, in time $O(mn)$. We store these values in a two dimensional array. We can now determine in constant time the value of $\varpi(\sigma+\alpha)$, for each stable equilibrium σ, and each modular reorientation of the jaw α.
The nodes and the edges of the graph based algorithm are encoded the same way as in Section 3.2. The graph, however, is built in a somewhat different way than before. A node of the graph again corresponds to a state of the part, encoded as an interval of possible orientations. The edges between the nodes are added a bit differently than before though. Let (I, t) be a node in the graph, I is an interval of stable equilibria and t is a fence type. Since each modular fence has a fixed angle, we can for *each* fence angle compute the image of I under the push function. If we add edges from (I, t) to the nodes in the graph corresponding to these images, Lemma 3.3 states that the resulting graph suffices to compute shortest paths in the graph, corresponding to shortest fence designs, although there might be redundant edges.
We compute the images as follows. For each modular fence f', having type t', we compute the reorientation of the jaw $\alpha_{t,f'}$ between a fence of type t, and the fence f'. The interval of orientations after appending fence f to a design is $I' = [\varpi(\sigma_i + \alpha_{t,f'}), \varpi(\sigma_j + \alpha_{t,f'})]$. We add a directed edge in the graph from

(I, t) to (I', t'). We label this edge with this modular push angle. For each node, we spend $O(m)$ time computing the outgoing edges.

Also, we add a source and a sink to the graph. We connect the source to every node (s, t) in the graph for which I contains all $|\Sigma|$ stable equilibria, and we connect every node (I, t) with I an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

The building time and the size of the graph are $O(mn^2)$. The search through the graph takes $O(mn^2)$ time as well. We can, therefore, compute a fence design in $O(mn^2)$ time, if one exists. If there exists no fence design which orients the part up to symmetry, then there is no path from the source to the sink in the graph, which is detected in $O(mn^2)$ time as well. If we assume that a constant number of fixed angles is available, then the running time is $O(n^2)$.

Theorem 3.14 *Let P be a polygonal part, with n vertices. Let m be the number of different available fence angles. If there exists a fence design that orients P, then a design of minimal length which orients this part can be computed in $O(mn^2)$. If no fence designs exists, the algorithm reports failure within the same time bound.*

We now present an output sensitive algorithm which computes a fence design in $O(mkn)$ time, with k being the number of fences. The algorithm is based on the output-sensitive algorithm of Section 3.3. If $k = o(n)$, this algorithm is asymptotically faster than the previously presented algorithm.

We store $O(n)$ intervals of possible orientations in an array. These intervals are the shortest intervals reachable with a fence design which is incrementally constructed. We have to take into account the fence type, and store therefore for each tuple of a stable orientation σ and a fence type t the shortest interval starting with orientation σ, with the last push by a fence of type t. When we augment the fence design with one fence, we can compute the new shortest intervals as follows. For each interval, we compute the m images of this interval. We check in the array of shortest intervals, if the new interval is shorter than the currently stored interval, and, if so, we replace the stored interval by the new interval. Looking up the image of an interval of stable orientations can be done in constant time, using our precomputed two-dimensional array with stored values of the push functions. Therefore, given the intervals after $k - 1$ fences, computing the intervals after k fences is accomplished in $O(nm)$ time.

We stop when the part is oriented up to symmetry. This can be checked by comparing the indices of the intervals of possible orientations. Unfortunately, not every part can be oriented up to symmetry using a modular fence design. This means we have to add an extra stop criterion which determines if the intervals keep improving, similar to the stop criterion for fence designs with friction we presented in Section 3.6. The following theorem summarizes the result.

Theorem 3.15 *Let P be a polygonal part, with n vertices. Let m be the number of different available fence angles. If there exists a fence design that orients P, then a design of minimal length which orients this part can be computed in $O(kmn)$ time, with $k = O(n^2)$ the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm reports failure in $O(mn^3)$ time.*

3.8 Implementation

We have implemented the algorithm described in Section 3.2 to test its behavior in practice. This turned out to be rather easy, using only some basic geometric computations for the push function, and some standard graph algorithms. We used the PlaGeo library for the geometric computations [44]. The resulting code is very fast; it returned fence designs within a fraction of a second for all parts we tried. All fence designs shown in this chapter were generated by the program. Our implementation offers the user the additional possibility of adding costs to graph edges. By doing so, the user can prevent the algorithm from outputting certain undesired types of fence designs. Assigning high costs to edges between any pair of nodes of the same fence type t, for example, will cause the algorithm to output a sequence of alternating (left and right) fences if such a sequence exists. Alternating sequences are often preferred over sequences containing cascades of left (or right) fences, as they generally allow for narrower conveyor belts (see Figure 3.8). Different cost assignments can be found to prevent e.g. unwanted steep and shallow fences. The costs make it impossible to apply Lemma 3.3 to reduce the graph size, as this may cause the removal of equally long or longer paths with lower cost from the graph. The size of the resulting graph is therefore $O(n^4)$. Dijkstra's algorithm (see e.g. [37]) has been used to find the minimal cost path through the graph in time $O(n^4)$.

3.9 Discussion

In this chapter we have investigated the problem of sensorless part orientation by sequences of fences. We have shown that any polygonal part can be oriented

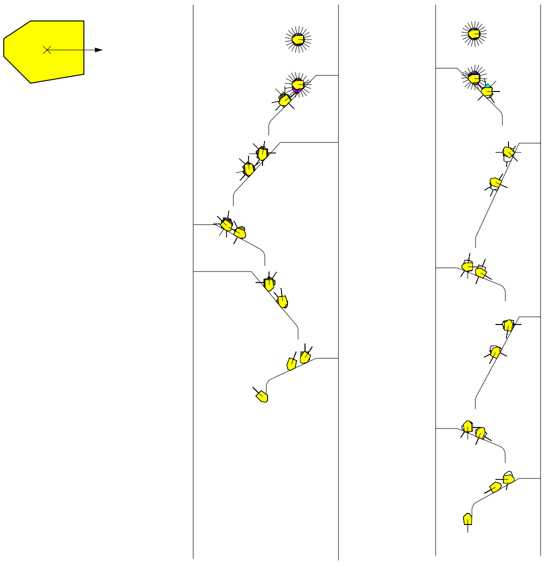


Figure 3.8 An optimal design of five fences, and a design of six alternating fences allowing for a narrower belt. Every line segment emanating from the part represents a possible orientation of the part.

up to symmetry by a sequence of fences placed along a conveyor belt. We have presented a polynomial-time algorithm for computing the shortest fence design for any given polygonal part. The algorithm is easy to implement and runs in time $O(n^3)$. The algorithm can be modified to compute fence designs which take into account various quality measures, such as avoidance of very steep or shallow fences. This modification increases the running time of the algorithm to $O(n^4)$. The structure of the algorithm yields an $O(n^2)$ bound on the length of the shortest fence design. We showed that for asymmetric parts the length is actually bounded by $O(n)$. It remains an open problem whether an $O(n)$ bound exists for parts that are not asymmetric.

Although pathological asymmetric polygons can be constructed that lead to push plans and fence designs of length $\Omega(n)$, it turns out that the length of most plans remains far below the worst-case length. Van der Stappen *et al.* [100] showed that only $O(1)$ actions are required for parts with non-zero eccentricity, i.e., with non-square minimum-width bounding box. The analysis also applies to curved parts, providing the first complexity bound for non-polygonal parts. The results generalize to fence designs for parts with acyclic left and right environments. It remains an open question whether this bound can be transferred to arbitrary polygonal, or curved parts.

We also presented an output-sensitive fence-design algorithm which benefits from the aforementioned bounds on the length of the plan. This algorithm runs in $O(kn \log n)$ time, with k the number of fences in the final design. Furthermore, we gave algorithms to compute fence designs for fences with friction and modular fences, in which we allow only a discrete set of fence angles. The former turned out to be just a minor modification to the algorithms for computing frictionless fence designs. The latter, however, gave rise to a different approach to computing fence designs in time $O(mn^2)$, or $O(mkn)$, where k denotes the number of fences, and m the number of fence angles. Unfortunately, not every part can be oriented with modular fences or fences with friction.

The results largely settle the algorithmic questions in computing fence designs, although some improvements in the running time might still be possible. The main open problem that is left is the question which parts can be oriented with friction or with modular fences. Experiments show that for many parts a valid fence design still exists. Another open problem is the bound on the length of the fence design. For general polygons, only an $O(n^2)$ upper bound is known at the moment. It is unknown whether this is indeed required or whether linear length designs always exist. Also, looking into orienting curved parts [87] using fences is of interest. Finally, we want to mention the problem of dealing with uncertainty in the shape of the part (see [31] for a first treatment). In Chapter 5 we give the generalization of fence design to three dimensions.

CHAPTER 4

Inside-out pulling

Picking up a part by grasping by e.g. a parallel jaw gripper is a common operation in industrial assembly. An advantage of grasping is, that the gripper eliminates little uncertainties in orientation of the part when it squeezes the part [66]. A range of slightly different orientations map all to the same orientation when the gripper is applied to the part [26]. Now, suppose that we have a part with an elevated edge, and a gripper that consists of two fingers, above the part. If we extend the gripper such that the two fingers squeeze the part inside out, we yield a grip on the part that is even more stable than the grip of the parallel jaw gripper: the inside-out grasp eliminates small uncertainties in the position as well as uncertainties in the orientation of the part.

Inspired by this observation, the question whether inside-out grasping can be used to orient a part from an unknown initial orientation, to a unique final orientation comes to mind. We present in this chapter a feeder that is related

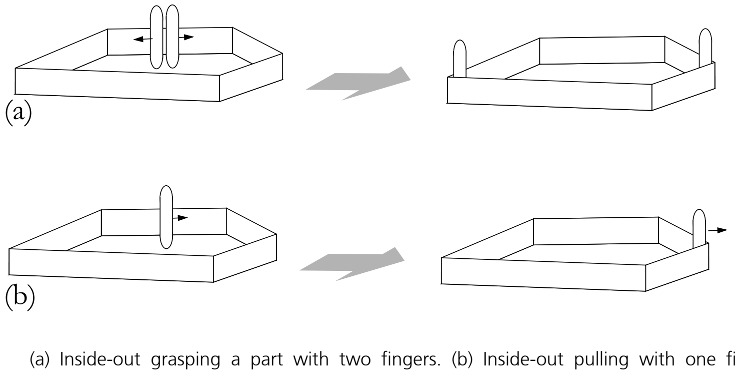


Figure 4.1 (a) Inside-out grasping a part with two fingers. (b) Inside-out pulling with one finger.

to inside-out grasping. Instead of two fingers, we consider one finger that manipulates the part by performing inside-out pull actions. The problem of pull planning is to design a sequence of pull actions (a *pull plan*) for the single pulling finger above a flat part with an elevated edge such that when applied to the part, it is moved to a unique orientation afterwards. Studying inside-out pulling from a geometric point of view builds a foundation in understanding the possibilities of orienting parts using inside-out grasping. A picture of an inside-out grasp on a part can be found in Figure 4.1(a). Figure 4.1(b) shows an inside-out pull action for the same part. Our goal is to design pull plans which have the feeding property. We will show that most convex polygonal parts can be oriented by a sequence of pull actions. In addition, we give an $O(n^3)$ algorithm for computing a sequence of pull operations for a given n-vertex convex polygonal part. We carefully analyze a basic pull action in a geometric framework. We prove several important properties of the finger within this framework. We derive a transfer function for the pulling feeder for convex polygonal parts, i.e. we define how the orientation of the part changes during a basic pull action. We shall show that this transfer function has several desirable properties. We show that, unfortunately, there exist non-convex polygonal parts that despite asymmetry cannot be fed using inside-out pull actions.

The outline of this chapter is as follows. We first discuss the pulling model in Section 4.1. In Section 4.2, we prove that most convex parts can be fed by the pulling feeder. In Section 4.3, we give the $O(n^3)$ algorithm to compute a sequence of pull operations to orient a given convex polygonal part. We discuss arbitrary (not necessarily convex) polygonal parts in Section 4.4, and show that the transfer functions for non-convex parts are not necessarily monotonic, and that there exist non-feedable parts. In Section 4.5, we conclude and pose several open questions.

4.1 Geometric modeling

In this section, we discuss the geometric properties of the pulling device. We address the problem in the plane. The pulling finger is assumed to be a frictionless point contact. Unless states otherwise, P denotes a convex 2-dimensional polygonal part. The number of vertices of P is given by n. It will turn out in Section 4.4 that there exist non-convex parts that cannot be oriented by a sequence of pull actions.

The part has a center-of-mass c, which lies inside the interior of the part. There is a fixed reference frame attached to P. Directions are specified relative to this frame. The *distance function* $\delta : [0, 2\pi) \rightarrow \mathbb{R}$ of P maps $\theta \in [0, 2\pi)$ onto the distance from c to the intersection of the boundary of P and the ray emanating from c in the direction of θ. Since P is convex, $\delta(\theta)$ is uniquely defined.

4.1.1 The effect of pulling a part

We consider the finger as a programmable part feeder. In Section 4.1.2, we will define a basic pull action and the corresponding transfer function. We first analyze the possible *configurations* of the finger with respect to the part. The configuration of the finger is a parametric representation of its initial position and pull direction. The finger has three degrees of freedom: the tuple $(x, y) \in \mathbb{R}^2$ specifying its position, and θ specifying its pull direction. The position and the direction are given relative to the reference frame of P. We denote the configuration of the finger by the triple (x, y, θ).

Assume for a moment that the finger is in contact with the boundary, and that the pull direction is away from the center-of-mass. In most cases, the part will start to rotate until the center-of-mass is collinear with the pull direction, and the contact point will slide towards a contact point with larger distance to the center-of-mass. If during pulling the contact point does not change, then contact point of the finger corresponds to either a local maximum of the distance function, or a (unstable) local minimum. We refer to the corresponding contact point as an equilibrium contact point. If the finger is at an *equilibrium contact* point, the pull direction is away from the center-of-mass, and the center-of-mass is on the supporting line of the pull direction through the contact point, then the finger is at an *equilibrium configuration*. An equilibrium contact point which corresponds to a local maximum in the distance function is called a *stable equilibrium* contact point, and the corresponding vertex of the part a *stable vertex*. Equilibrium contact points which correspond to local minima in the distance function are called *unstable equilibrium* contact points.

Let m denote the number of stable vertices of P. We denote the stable vertices of P by $v_1, \ldots, v_m$. The stable vertices are numbered in counterclockwise order along the boundary of P. The unstable contact point between two successive vertices v_i and v_{i+1} is denoted by $u_{i,i+1}$. Indexing is modulo the number of stable vertices. In Figure 4.2 an overview of possible configurations of the finger is given. In the figure, the stable vertices and the unstable contact points are enumerated. The part has one unstable vertex (which is not enumerated). There are dotted lines emanating from the center-of-mass through the unstable contact points. The following configurations can be found in the picture:

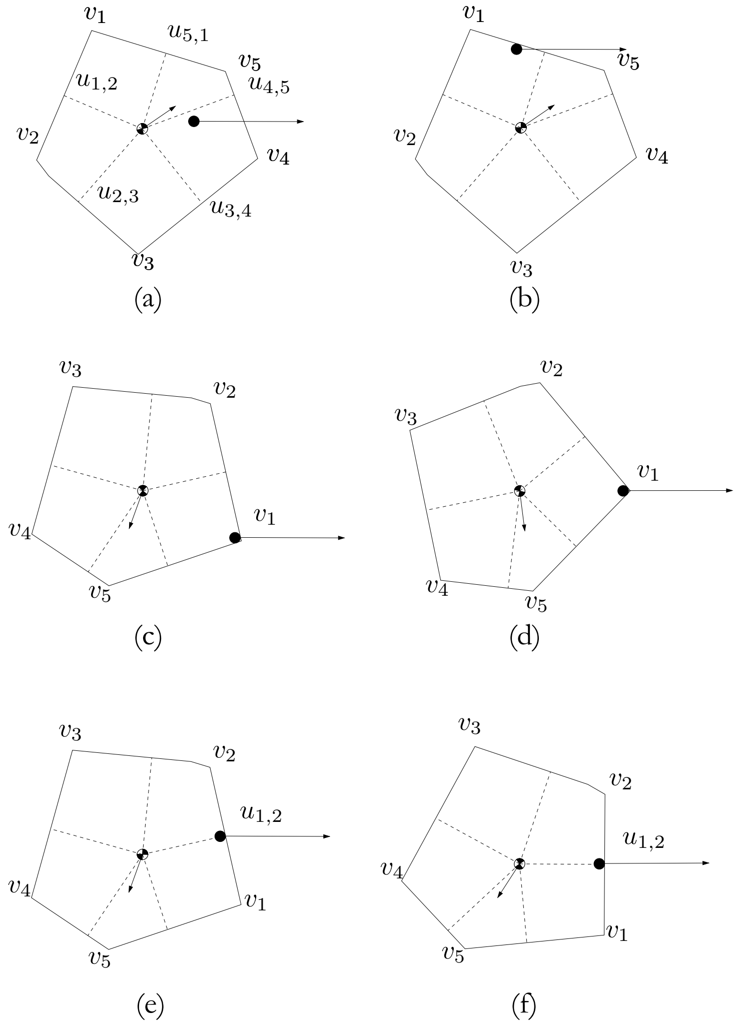


Figure 4.2 An overview of possible configurations.

a. A configuration for which the finger is in the interior of the part.
b. A configuration of the finger at the boundary of the part.
c. A configuration of the finger at a stable vertex of the part.
d. An equilibrium configuration at a stable vertex.
e. A configuration at an unstable contact point.
f. An equilibrium configuration at an unstable contact point.

There are infinitely many different configurations of the finger. If we pull along a straight line, we will first reach the boundary, and then the part will start to rotate

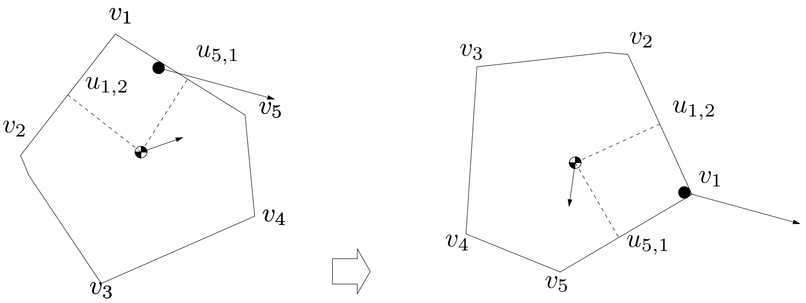


Figure 4.3 Pulling a part. The distance between c and the finger strictly increases and, after alignment, the finger is in contact with vertex v_1.

unless the finger is in a stable configuration. We assume that, continuing from such a configuration, the distance between the point of contact at the boundary of the part and the part's center-of-mass only increases during the rotation of the part. This assumption is valid for parts with the distribution of friction concentrated at the center-of-mass (and the instantaneous center-of-rotation coincides with the center-of-mass), and zero friction between the boundary and the finger. As a consequence, the angle between the pull direction and the edge of the part does not influence the final orientation of the part. We subdivide the boundary of the part into regions for which the final vertex is the same. Any interval on the boundary between two successive unstable contact points map onto the stable vertex between the contact points. An unstable contact point maps onto an unstable equilibrium configuration. In Figure 4.3, we see a part before and after a pull action. The first intuition might be that the finger will reach vertex v_5; the distance function dictates that the final vertex is v_1, though. After a pull action starting in an arbitrary configuration, we eventually yield an equilibrium configuration which is either a stable equilibrium configuration at a stable vertex of P or an unstable equilibrium configuration at an unstable contact point. It is easy to see that there are $O(n)$ stable configurations.

4.1.2 The pull function

In this section, we define a *basic pull action*, which maps a equilibrium configuration of the finger onto another equilibrium configuration for a given pull direction. Hence, similar to the basic push action [26,45], the basic pull action serves as a basis for a transfer function for a part feeder which uses pull operations to orient parts to a unique final orientation.

We denote the angle of the ray emanating from c through v_i by σ_i. In other words, σ_i links the orientation of P relative to its reference orientation to stable equilibrium configuration at vertex v_i. Similary, we denote the angle of the ray from c through the unstable contact point $u_{i,i+1}$ by $\sigma_{i,i+1}$. We assume that the finger is in contact with vertex v_i of P, and the configuration of the finger is a stable equilibrium configuration. A basic pull action moves the finger along a straight line, starting at v_i, until the finger is again in a equilibrium configuration, i.e. we pull long enough to be certain that the part has stabilized. We express the pull direction for the basic pull action relatively to the current pull direction, which is along the ray emanating from c, in the direction of σ_i, through v_i.

We first consider pull directions which initially direct the finger into the interior of the part. In this case, the finger will reach a configuration on the boundary of the part different from the initial contact, and the part will start to rotate according to the assumptions.

The subdivision of the boundary of P into regions which map onto different stable vertices of P induces a subdivision of the possible pull directions from v_i. If move the pulling finger, starting in vertex v_i, it will (for most pull directions) first hit the boundary at the intersection with the ray emanating from v_i in the pull direction. Let $\tau_{v_i} : [0, 2\pi) \to [0, 2\pi)$ be the function which maps a pull direction from v onto the aforementioned intersection. For directions θ for which this intersection is undefined, $\tau_{v_i}(\theta) = \sigma_i$. See Figure 4.4 for a picture of a part and $\tau_{v_1}(\theta)$ for pull direction θ from v_1. Let $l_{v_i}(v_j)$ be the pull direction from v_j such that $\tau_{v_i}(l_{v_i}(v_j))$ intersects $u_{j-1,j}$, and $r_{v_i}(v_j)$ be the direction such that $\tau_{v_i}(r_{v_i}(v_j))$ intersects $u_{j,j+1}$. Pull directions in the angular interval $(l_{v_i}(v_j), r_{v_i}(v_j))$ will eventually reach a stable configuration of the finger at stable vertex v_j.

The *pull function* $\varpi_{v_i} : [-\pi, \pi) \to [0, 2\pi)$ for vertex v_i links a pull direction from vertex v_i onto the orientation of the part relatively to the pull direction after the completion of the basic pull action. See Figure 4.5 for a picture of a pull function. Like the push function [45], the pull function is a step function. The steps map onto stable equilibrium contact vertices, points separating the steps map onto unstable contacts. Similarly, we can define the pull function for an unstable equilibrium of P.

4.1.3 Properties of the pull functions

Throughout the remainder of this chapter, we focus on the pull function for stable vertices of P. This is not a real restriction, since it is always possible to

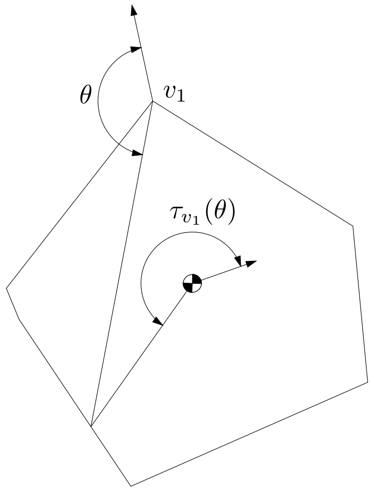


Figure 4.4 A part with $\tau_{v_1}(\theta)$ for the given pull direction θ from vertex v_1.

transfer any unstable equilibrium of P onto a neighboring stable equilibrium by pulling in direction $\pi/2+\epsilon$. Such a pull direction would map a stable equilibrium of P unchanged back onto itself.

The most important property of the collection of pull functions (of stable equilbria of P) we prove in this section is monotonicity, or the order preserving property [76]. We recall from Section 2.2.1 that a sequence $v_1, \ldots, v_n$ of stable vertices is *ordered* if $v_1, \ldots, v_n$ are encountered in order when the boundary of P is traced *once*, starting from v_1. The set pull functions is *monotonic* if for any pull direction θ and any ordered sequence $v_1, \ldots, v_n$ the sequence $\varpi_{v_1}(\theta), \ldots, \varpi_{v_n}(\theta)$ is also ordered.

We need some geometric properties of the pull function and the part. In order to derive bounds on the part shape related to the pull function, we use the elementary trigonometric result that the intersection of any pair of lines through two points p and q that intersect with angle ϕ lies on a circle determined by p, q and ϕ. Figure 4.6 shows an example of the circle. We state this property in Lemma 4.1.

Lemma 4.1 *Let p, q be two points in $\mathbb{R}^2$. Let ℓ_p and ℓ'_p be two distinct lines through p. Let ℓ_q and ℓ'_q be two lines through q respectively. Let s and s' be the intersections of ℓ_p and ℓ_q and ℓ'_p and ℓ'_q respectively. If $\angle psq = \angle ps'q$ and s and s' lie in a single halfplane bounded by the line through p and q, then, p, q, s and s' are cocircular.*

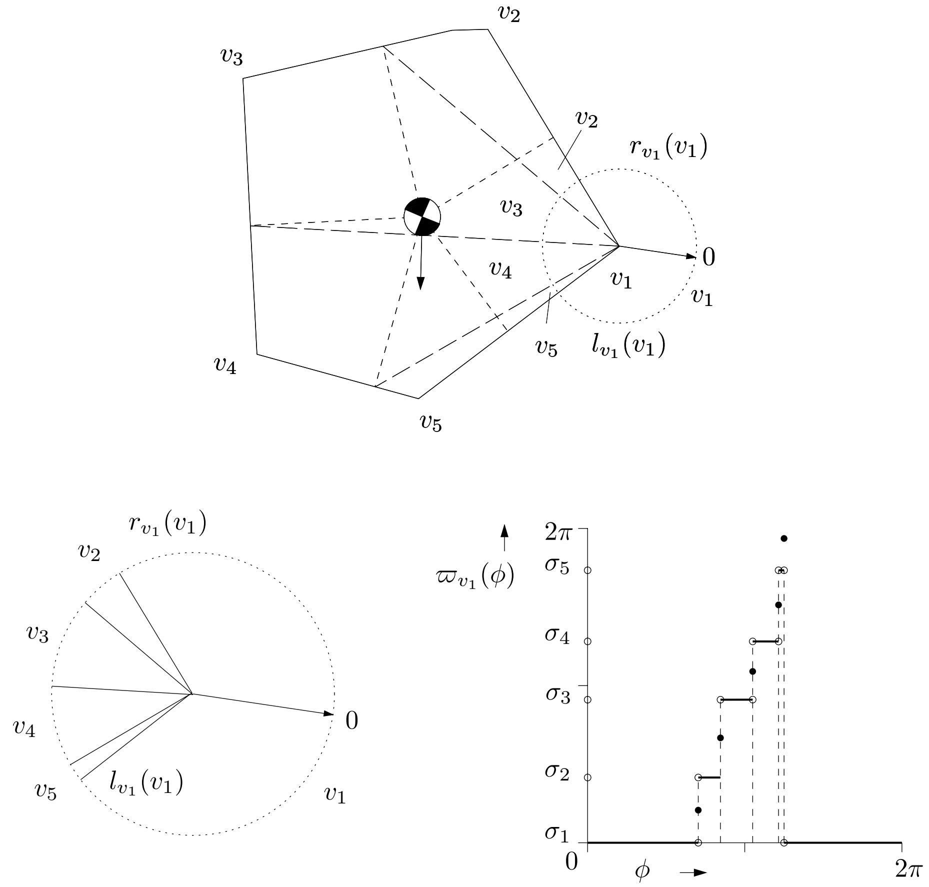


Figure 4.5 The derivation of the pull function for vertex v_1 of the depicted part. First, we determine the pull directions that map onto unstable contact points. This gives a subdivision of the possible pull directions that translates into the pull function for v_1.

The next lemma gives a useful restriction on the relative positions of the stable vertices of P.

Lemma 4.2 *Let P be a convex polygonal part with center-of-mass c. Let v_i be a stable vertex of P. Let $\mathcal{C}_e$ be the circle with diagonal (c, v_i). There are no stable vertices of P in the interior of $\mathcal{C}_e$.*

Proof: Any stable vertex other than v_i must be separated from v_i by an unstable equilibrium contact. An unstable equilibrium contact corresponds to a local

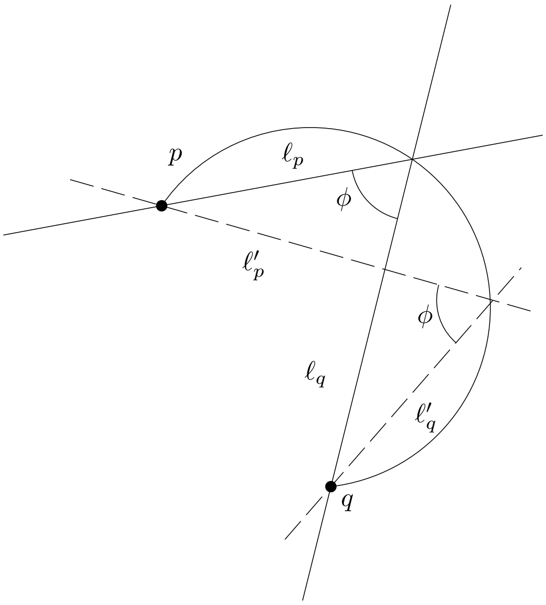


Figure 4.6 Intersections of lines through p and q on a circle.

minimum in the distance function. Hence, at the equilibrium contact, the ray emanating from c and intersecting the contact is orthogonal to an edge of P. See Figure 4.7. Using Lemma 4.1 we derive that the intersection of the edge neighboring v_i and the ray from c define circle $\mathcal{C}_e$. Since P is convex, and c must be in the interior of P, the closest unstable equilibrium contact point is on an edge connected to v_i. The possible contact points on an edge connected to v_i lie on $\mathcal{C}_e$. Hence a stable equilibrium contact point lies outside $\mathcal{C}_e$. □

Next, we give a weak order-preserving lemma.

Lemma 4.3 *Let v_a and v_b be distinct vertices of P. Let ϖ_{v_a} be the pull function from vertex v_a, and ϖ_{v_b} be the pull function from vertex v_b. For any pull direction θ in $(0, r_{v_a}(v_a))$, for which $\varpi_{v_a}(\theta) = \sigma_a$, the sequence $\sigma_b, \varpi_{v_b}(\theta), \sigma_a$ is in counterclockwise order.*

Proof: The proof will be by contradiction. By construction, σ_a, $\sigma_{a,a+1}$, σ_b are in counterclockwise order. Suppose that there is a pull direction $\theta \in (0, r_{v_a}(v_a))$ such that $\sigma_a, \varpi_{v_b}(\theta), \sigma_b$ are in counterclockwise order.

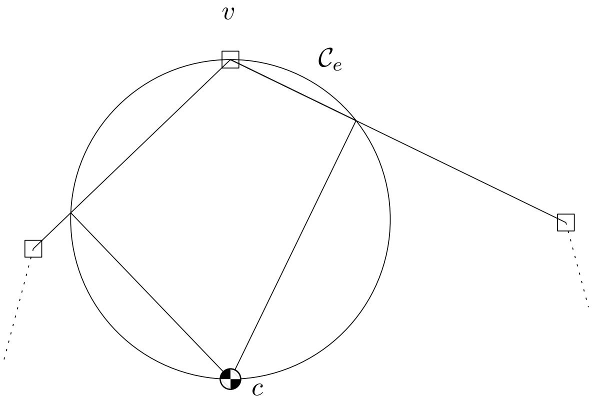


Figure 4.7 Any stable vertex must lie outside the circle with diagonal (c, v).

We focus on the rays with pull direction θ emanating from v_a and v_b. Pull directions are expressed relatively to the zero direction of v_a and v_b respectively. Hence, the rays for pull direction π both intersect the center-of-mass of P. Using Lemma 4.1 the intersections of the pull direction rays lie on a circle $\mathcal{C}_r$ through v_a, v_b and c—see Figure 4.1.3.
From Lemma 4.2 it follows that the vertices of P lie outside circle with diagonal (c, v). $\mathcal{C}_e$ through v_a and c. As a consequence, for any intersection of the ray of a pull direction emanating from v_b with any preimage of orientations between σ_a and σ_b, the order is σ_a, $\tau_{v_b}(\theta)$, $\tau_{v_a}(\theta)$, σ_b. For such pull directions, $\varpi_{v_a}(\theta) \neq \sigma_a$, and the theorem follows by contradiction. □

Lemma 4.4 *Let v_a and v_b be distinct vertices of P. For any pull direction $\theta \in [0, 2\pi)$, $[\tau_{v_a}(0), \tau_{v_a}(\theta)] \nsubseteq [\tau_{v_b}(0), \tau_{v_b}(\theta)]$*

Proof: Suppose that $\tau_{v_a}(0) \in [\tau_{v_b}(0), \tau_{v_b}(\theta)]$. We show that $\tau_{v_a}(\theta) \notin [\tau_{v_b}(0), \tau_{v_b}(\theta)]$. See Figure 4.1.3. We show that the parameterized boundary points, which coincide with the intersection of rays from the vertices v_a and v_b with the boundary of P have the correct order.
The rays of pull direction π both intersect the center-of-mass of P. Using Lemma 4.1, we derive that the intersections of corresponding rays for the pull directions lie on a circle $\mathcal{C}_r$ through v_a, v_b and c.
As a consequence, for any intersection of the ray emanating from v_b between any preimage of vertices between v_a and v_b, the order is σ_a, $\tau_{v_b}(\theta))$, $\tau_{v_a}(\theta))$, σ_b. □

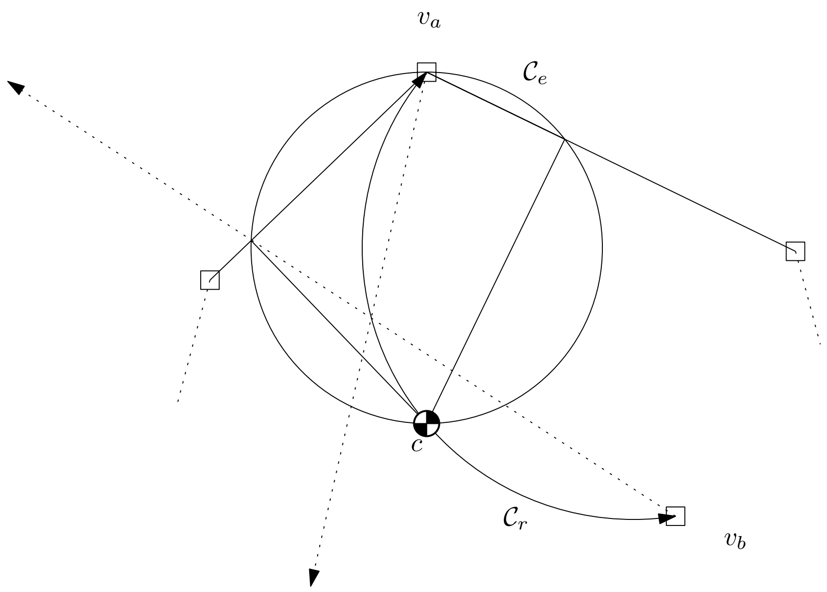


Figure 4.8 Illustration of Lemma 4.3.

Theorem 4.5 *Let* P *be a convex polygonal part. Let* $v_1, \ldots, v_n$ *be the stable vertices of* P*. Let* $\varpi_{v_1}(\cdot), \ldots, \varpi_{v_m}(\cdot)$ *be the set of pull functions of* P*. The set of pull functions is monotonic.*

Proof: Let $v_a, v_b, v_c, \ldots$ be ordered counterclockwisely. We have to show that for any direction θ, the sequence $\varpi_{v_a}(\theta), \varpi_{v_b}(\theta), \ldots$ is also ordered. We shall show that any triple $\varpi_{v_a}(\theta), \varpi_{v_b}(\theta), \varpi_{v_c}(\theta)$, is ordered counterclockwisely. Let us assume, for a contradiction, that there is a triple that is not ordered counterclockwisely, but clockwisely instead.
From Lemma 4.4 it follows that $[\tau_{v_a}(0), \tau_{v_a}(\theta)] \not\subseteq [\tau_{v_b}(0), \tau_{v_b}(\theta)]$. Hence, either $[\tau_{v_a}(0), \tau_{v_a}(\theta)]$ and $[\tau_{v_b}(0), \tau_{v_b}(\theta)]$ are disjoint or partially overlap.
Vertex v_c is between v_b and v_a in the counterclockwise order of the vertices. Hence $\tau_{v_c}(0) \in [\tau_{v_b}(0), \tau_{v_a}(0)]$. In order for $[\tau_{v_c}(0), \tau_{v_c}(\theta)]$ to not contain $[\tau_{v_a}(0), \tau_{v_a}(\theta)]$, we must have $\tau_{v_c}(\theta) \in [\tau_{v_c}(0), \tau_{v_a}(\theta)]$, but this violates either the counterclockwise order of $\varpi_{v_a}(\theta), \varpi_{v_c}(\theta), \varpi_{v_b}(\theta)$, or the (partially) disjointness of $[\tau_{v_a}(0), \tau_{v_a}(\theta)]$ and $[\tau_{v_b}(0), \tau_{v_b}(\theta)]$.
Hence, the clockwisely ordered triple does not exist and it follows that the pull function is monotonic by contradiction. $\square$

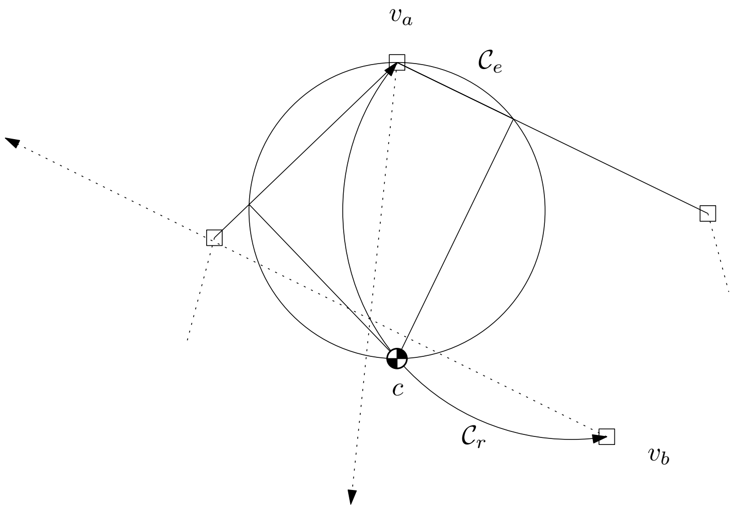


Figure 4.9 Illustration of Lemma 4.4.

4.2 Completeness of pulling

In Section 2.2.2 we showed how to orient polygonal parts by means of push operations. These push plans are originally due to Chen and Ierardi [32]. Although their plans are not always the shortest push plans possible, they give an upper bound on the length of their plans which bounds the number of necessary push operations by $O(n)$. Moreover, from their analysis follows easily that a push plan always exists, unless the part has symmetry—still, the part can be oriented up to its symmetry in that case. In this section, we shall use a similar strategy to show that most convex polygonal parts can be oriented up to symmetry using a linear number of pull operations.

For any vertex v_i a pull direction in the angular interval $(l_{v_i}(v_i), r_{v_i}(v_i))$ maps v_i back onto itself. Let λ denote the maximum of $r_{v_i}(v_i)$ for $i \in [1, \ldots, m]$. We focus on the case that there is unique vertex v^* for which $\lambda = r_{v^*}(v^*)$. Let ϵ be a small positive constant, such that $\lambda - \epsilon$ is greater than $r_{v_i}(v_i)$ for all $v_i \neq v^*$. We have the following simple pull plan for orienting the given polygon.

```
1. for each $i = 1$ to $n - 1$ do
    1.1 pull in direction $\lambda - \epsilon$ until the finger is at a stable configuration
```

Without loss of generality, we assume that $r_{v_m}(v_m) = \lambda$. We will show that this algorithm will finally reach a configuration for which the finger is in contact

with vertex v_m. Initially, $v_1, \ldots, v_m$ is the (counterclockwisely) ordered sequence containing all possible stable vertices of P. We shall show that a basic pull action of the above pull plan will map an ordered sequence of possible stable vertices $v_i, \ldots, v_m$ onto an ordered sequence contained in $v_{i+1}, \ldots v_m$.
Firstly, we observe that the change of the pull direction is such that vertex v_m is mapped onto itself. Secondly, we observe that any vertex $v_{\bar{\imath}}$, $i \leq \bar{\imath} < m$ will *not* map $v_{\bar{\imath}}$ onto itself. From Lemma 4.3 it follows that $v_{\bar{\imath}}$ is mapped onto a vertex in $v_{i+1}, \ldots, v_m$. Hence, every pull action reduces the number of possible orientations by at least one, and the following theorem follows.

Lemma 4.6 *Let P be a convex polygonal part with m stable vertices. Let λ denote the maximum of $r_{v_i}(v_i)$ for $i \in [1, \ldots, m]$. If there is a unique vertex v^* for which $r_{v^*}(v^*) = \lambda$, then we can orient P using $O(n)$ pull actions.*

If we drop the assumption that there is a unique vertex v^* for which $\lambda = |r_{v^*}(v^*)|$, the simple pull plan does not orient the part. The reader might recall that for push planning, we can apply Lemma 2.6 (the Stretching Lemma), and find a linear plan to orient the part. It would be nice to have a similar lemma for pull planning. Unfortunately, the stretching lemma does not easily carry over to pull plans. We conjecture, though, that it is possible to find a similar lemma for pull plans, and as a consequence to find a linear length pull plan for any convex polygonal part with asymmetric pull functions.

4.3 Computing optimal pull plans

In this section, we outline how to compute a sequence of pull operations to orient P. We follow an approach similar to that for fence design in Section 3.2. The strategy of the algorithm for computing pull plans is based on reducing uncertainty. We call a set of possible orientations of the part a *state* of the part. The goal is to find basic actions which map a state of the part onto a state of smaller cardinality. By concatenating such basic actions we aim to finally reach a state corresponding to a single orientation of P.
The algorithm we propose is based on a graph search. We encode states of the finger as nodes of a graph, and a directed edge between a pair of nodes if there exists a pull direction which maps the state of the former onto the latter. From the monotonicity of the pull functions it follows that we do not need a node for every set of possible orientations of P. We can suffice with nodes for every interval of possible orientations of P. Hence, we have $O(n^2)$ nodes, one for each

pairs of stable vertices of P. Theorem 3.5 shows that there exists a graph of size $O(n^3)$ in which a shortest path corresponds to a shortest pull plan.

We construct the graph in a similar way as the graph for fence design, which is rather simple. In the graph, each node with interval $[v_i, v_j]$, has just one (outgoing) edge to the set of nodes with intervals that have a common left endpoint $v_{i'}$. This outgoing edge connects the former node to the node $v_{i'}, v_{j'}$, reachable by a single pull action from v_i, such that the corresponding interval of orientations, $[\sigma_{i'}, \sigma_{j'}]$, is minimized in length.

The shortest interval with left endpoint $\sigma_{i'}$ is derived by a pull direction which maps v_i onto $l_{v_i}(v_{i'})$. From this observation, the construction of the graph follows automatically. We align the interval with the left environment of the reachable orientations for a valid reorientation of the pull direction, and compute the resulting interval after application of the pull function. Hence, computing all outgoing edges for one node can be carried out in linear time.

A shortest path from the source to the sink of the graph corresponds to a shortest pull plan to orient P. We can find such a path in linear time in the size of the graph, or report that no such path exists within the same time bound. The next theorem summarizes this section.

Theorem 4.7 *Let P be a convex polygonal part with n vertices. In $O(n^3)$ time we can construct a pull plan which orients P, if such a plan exists. Otherwise, we can report failure within the same time bound.*

It is not hard to modify the output-sensitive algorithm for computing fence designs to an output-sensitive algorithm for pull plans. The running time of the resulting output-sensitive algorithm for pull plans is similar to the running time of the output-sensitive algorithm for fence design. The problem is that in the case of an arbitrary part, we do not know in advance whether a pull plan exists, and we can only stop after extending the plan to $\Omega(n^2)$ basic pull actions.

4.4 Arbitrary polygonal parts

In the case of an arbitrary polygonal part, the pull function needs no longer be monotonic. Figure 4.10(a) shows an example of such a part. For pull direction π, the ordered sequence of stable vertices v_1, v_2, v_3, v_4 is mapped onto the sequence $\sigma_3, \sigma_2, \sigma_1, \sigma_2$ (corresponding to vertices v_3, v_2, v_1, v_2) which is not ordered. This observation has impact on both the completeness and the algorithmic complexity

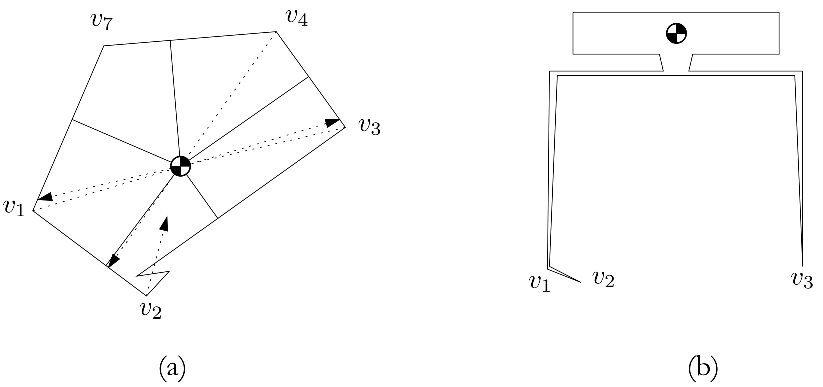


Figure 4.10 (a) A part with non-monotonic pull functions. (b) An unfeedable part.

of pulling non-convex parts. On the algorithmic side, we can no longer use our graph based algorithm of Section 4.3.

Eppstein [40] presented a general algorithm for part feeders with non-monotonic transfer functions. The algorithm outputs a plan, if such a plan exists, but is not guaranteed to give the optimal plan in terms of its length. We can compute a pull plan for an arbitrary polygonal part, if such a plan exists, by means of Eppstein's algorithm for part feeders with non-monotonic transfer functions. The algorithm runs in $O(n^6)$ time in our case, and is not guaranteed to give the shortest pull plans. Eppstein showed that finding a shortest plan for a general part feeder with non-monotonic transfer functions is NP-complete. It is an open question whether his bound applies to pull plans, or to find an algorithm for computing a shortest pull plan that runs in polynomial time.

We can construct non-convex parts which cannot be fed by our finger. In Figure 4.10(b) we show such an unfeedable part. The pull function for vertex v_1 maps to σ_1, or σ_2. The pull function for vertex v_2 maps to σ_2 or σ_1. The pull function for vertex v_3 maps to σ_3 for any pull direction. Hence, if the part is in a stable configuration at v_1 it will never be able reach an orientation corresponding to v_3 and vice versa. Hence there is no unique final configuration of the part that can be reached from any initial configuration.

4.5 Discussion

In this chapter we investigated the problem of sensorless part orientation by sequences of inside-out pull actions. We showed that almost any convex polygonal part can be oriented by a sequence of pull actions. We presented a polynomial-time algorithm for computing the shortest pull plan for a given polygonal part, if such a plan exists. The structure of the algorithm yields an $O(n^2)$ bound on the length of the shortest pull plan. We showed that for asymmetric parts which have a unique vertex v^* for which $r_{v^*}(v^*)$ (or $-l_{v^*}(v^*)$) is maximal, the length of a pull plan is bounded by $O(n)$. It remains an open problem whether pull plans exist for parts that are not asymmetric, and whether the bound on the length of a pull plan of $O(n)$ is applicable to this class of parts. In the case of several orientations v for which $r_v(v)$, or $-l_v(v)$) is maximal, we cannot apply Lemmas 2.5 and 2.6 to deal with them (as in the case of pushing), because the pull function of one vertex is not a shifted version of the pull function of another vertex.

The pull function for vertex v is computed from τ_v. It is not hard to see that the pull function for a distinct vertex is not the simple shifted pull function of v. We believe, however, that it is possible to derive a similar lemma for pull functions, based on the geometry of P. Such a lemma would immediately show that any part with distinct pull functions can be oriented, by repeatedly finding shorter intervals of orientations.

The next step towards a stretching lemma for pull plans would be to show how to get back on the track of the simple pull plan of Section 4.2 using a constant number of basic pull actions. Again, we have to take into account the distinct pull functions for the vertices of P. We conjecture that a stretching lemma for pull plans exists. If the conjecture turns out to be false, it is an open problem to characterise the clas of parts feedable by pulling. The structure of fence designs for arbitrary parts can also be applied to generate pull plans, and will most likely feed any part without rotational symmetry. There seem to be some technicalities to adapt the case of the double cyclic push functions. Unfortunately, such a strategy for pull plans would only prove completeness, and not a linear upperbound on the length of the resulting plans.

We presented a graph based algorithm to compute pull plans. One can also opt to use an output-sensitive algorithm to compute pull plans. The output-sensitive algorithm maintains for each stable orientation σ of P, the shortest interval of possible orientations that starts with σ after k basic actions. From an algorithmic point of view, it is even more challenging to find out whether we can suffice with a greedy algorithm (which only maintains one interval of possible orientations of P) for computing pull plans, similar to Goldberg's algorithm for push planning [45]. If such a strategy is justified, we would get time bounds for pull planning similar to the bounds for push planning, i.e. $O(n^2)$ if we can show that pull plans have linear length, or $O(n^2 \log n)$ otherwise [45].

Although pathological polygons can be constructed that lead to push and pull plans of length $\Omega(n)$, it turns out that the length of most plans remains far below the worst-case length for push plans. Van der Stappen *et al.* [100] showed that only $O(1)$ actions are required for parts with non-zero eccentricity, i.e. with non-square minimum-width bounding box. We expect that a similar analysis can be carried out for pull plans.

In the case of arbitrary (not necessarily convex) parts, we are able to construct parts that do not have monotonic transfer functions. We are even able to construct parts that cannot be fed by a sequence of pull actions. Eppstein [40] showed that finding a shortest plans is NP-complete for part feeders that have general (non-monotonic) transfer functions. It is an open question whether we can find an algorithm for computing a shortest pull plan that runs in polynomial time. Also, we still seek a geometrical characterization of parts that can be fed by the finger.

If we extend inside-out pulling to inside-out grasping, we have to take into account one extra degree of freedom of the device: the width of the gripper. In the line of thought of this chapter, we would like to determine a discrete subset of configurations, and a basic action which defines a transfer function for the subset of configurations. A possible subset of configurations is the set of configurations for which the width of the gripper is a local maximum. A way of extending inside-out pulling to inside-out grasping is to define the basic action in which we first pull the part with with a single finger, and subsequently extend the gripper. Such a *pull-squeeze* action, similar to the push-squeeze action [45], is perhaps the most straightforward extension of pull planning. Another way of defining a basic action is to only allow the gripper to extend and reorient. Similar to squeezing with a parallel jaw gripper, the first step in analyzing this basic action could be to consider the degenerate case in which both fingers of the gripper touch the part simultaneously, and there is no pull phase. For any basic action for inside-out grasping, we would like to show that the corresponding transfer function is monotonic. Moreover, we would like to address completeness issues and find algorithms to design inside-out grasp plans.

CHAPTER 5

Pushing into the third dimension

The drawback of the majority of the achievements in the field of sensorless orientation is that they only apply to flat, two-dimensional parts, or to parts where the face the part rests on is known beforehand. In this chapter, we narrow the gap between industrial feeders and the scientific work on sensorless orientation, by introducing a feeder which orients three-dimensional parts up to symmetry. This is the first device which can be proven to correctly feed three-dimensional parts. The device we use is a cylinder with plates tilted toward the interior of

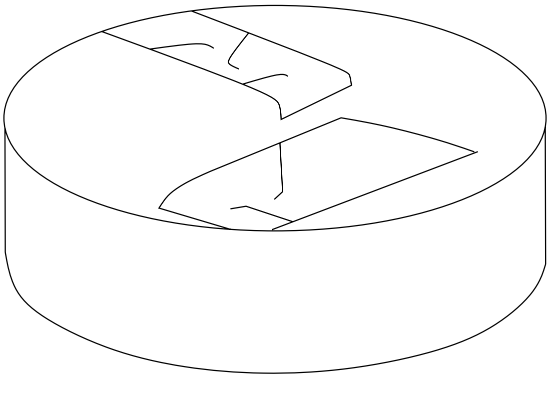


Figure 5.1 The three-dimensional part feeder. Plates with fences mounted to a cylinder.

the cylinder attached to the side. Across the plates, there are fences. The part cascades down from plate to plate, and slides along the fences as it travels down a plate. A picture of the feeder is given in Figure 5.1. The goal of this chapter is to compute the set-up of plates and fences that is guaranteed to move a given asymmetric polyhedral part towards a unique final orientation. Such a set-up, consisting of a sequence of plate slopes, and for each plate a sequence of fence orientations is referred to as a *(plate and fence) design*.

When a part moves along a plate and touches a fence on it, it is in some sense pushed from two different, orthogonal directions: the plate and the fence. This motivates us to first study the artificial problem of pushing in three-dimensional space. Here, the part is assumed to float in the air while we push it from two orthogonal directions. We show that a three-dimensional polyhedral part P can be oriented up to symmetry by a (particular) sequence of push actions, a *push plan*, of length $O(n^2)$, where n is the number of vertices of P. Furthermore, we give an $O(n^3 \log n)$ time algorithm to compute such a push plan. We show how to transform this three-dimensional push plan to a three-dimensional design for the plates and fences. The resulting design consists of $O(n^3)$ plates and fences, and can be computed in $O(n^4 \log n)$ time.

The chapter is organized as follows. We first discuss the device we use to orient parts, introduce the corresponding pushing jaw, and study the behavior of a part being pushed in Section 5.1. We then show, in Section 5.2, that the jaw can orient any given polyhedral part up to symmetry. In Section 5.3 we show how to compute a sequence of push actions to orient a given part. In Section 5.4 we show how the results for the generic jaw carry over to the cylinder with plates and fences. In Section 5.5, we conclude and pose several open problems.

5.1 Pushing parts

A polyhedral part in three-dimensional space has three rotational degrees of freedom. There are numerous ways to represent orientations and rotations of objects in the three-dimensional world. We assume that a fixed reference frame is attached to P. We denote the orientation of P relative to this reference frame by $\sigma = (\phi, \psi, \theta)$, where (ϕ, ψ) are the polar coordinates of a point on the sphere of directions, and θ the roll, which is a rotation about the ray emanating from origin, intersecting (ϕ, ψ). See Figure 5.2 for a picture. This representation will be shown to be appropriate considering the rotational behavior of the part as it aligns to our feeder. We discuss our feeder in Section 5.1.1. The rotational behavior of P in contact with the feeder is discussed in Section 5.1.2.

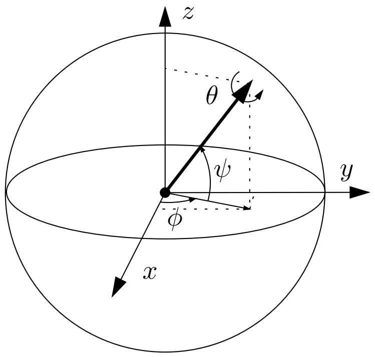


Figure 5.2 The rotation is specified by a point (ϕ, ψ) on the sphere of directions, and a rotation θ about the vector through this point.

5.1.1 Modeling the feeder

A part in three-dimensional space can have infinitely many orientations. The device we use to orient this part discretizes the set of possible orientations of the part. The feeder consists of a series of bent plates along which the part cascades down. Across a plate, there are fences which brush against the part as it slides down the plate. A picture of a part sliding down a plate is given in Figure 5.3(a). The plate on which the part slides discretizes the first two degrees of freedom of rotation of the part. A part in alignment with a plate retains one undiscretized rotational degree of freedom. The rotation of the part is determined up to its roll, i.e. the rotation about the axis perpendicular to the plate. The fences, which are mounted across the plates, push the part from the side, and discretize the roll of its rotation. We assume that P first settles on the plate before it reaches the fences which are mounted across the plate, and there is only rotation about the roll axis as the fences brush the part.

We look at the push actions of the plates and the fences in a more general setting. We generalize the cylindrical feeder by substituting the plate along which the part slides by a plane on which the part rests. We substitute the fences by an orthogonal second plane, which pushes the part from the side. We call the planes the primary and secondary (pushing) plane, respectively. A picture of the resulting jaw is given in Figure 5.3(b).

Since the planes can only touch P at its convex hull, we assume without loss of generality that P is convex. We assume that the center-of-mass of P, denoted by c, is inside the interior of P. Analogously to the cylindrical feeder, we assume that only after P has aligned with the primary plane, we apply the secondary

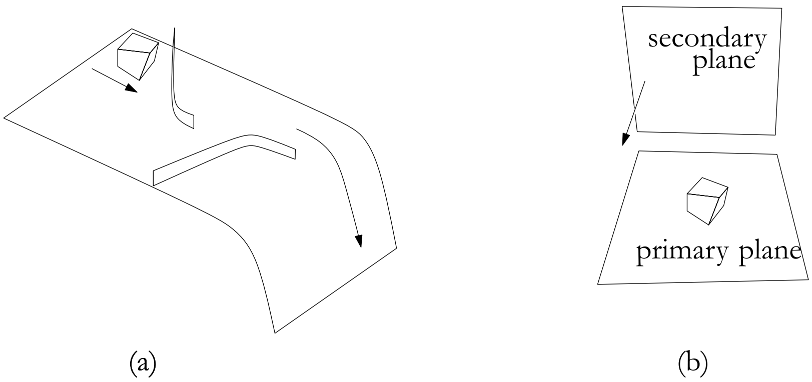


Figure 5.3 (a) A part sliding down a plate with fences. (b) The same part on the jaw.

plane. As the part rests on the primary plane, the secondary plane pushes P at its orthogonal projection onto the primary plane. We assume that the feature on which P rests retains contact with the primary plane as the secondary plane touches P. We assume that for any equilibrium orientation, which is an orientation for which P rests on the jaw (see Section 5.1.2 for a definition of an equilibrium orientation), the projection of P onto the primary plane has no (periodic) symmetry. We refer to a part with this property as being *asymmetric*.

In order to be able to approach the part from any direction, we make the (obviously unrealistic) assumption that the part floats in the air, and assume that we can control some kind of gravitational field which attracts the part in a direction towards the jaw. Also, we assume that the part quasi-statically aligns with the jaw, i.e. we ignore inertia. Studying this unrealistic situation is useful for analyzing our feeder later.

In order to be able to determine a sequence of push directions that orients P, we need to understand the rotational behavior of P when pushed by the jaw. We analyze this behavior below.

5.1.2 The push function

A basic action of the jaw consists of directing and applying the jaw. The result of a basic action for a part in its reference orientation is given by the *push function*. The push function $\varpi : [0, 2\pi) \times [-\pi/2, \pi/2] \times [0, 2\pi) \to [0, 2\pi) \times [-\pi/2, \pi/2] \times [0, 2\pi)$ maps a push direction of the jaw relative to P in its reference orientation onto the orientation of P after alignment with the jaw. The orientation of P after a basic action for a different initial orientation than its reference orientation is equal to the push function for the push direction plus the offset between the reference and the actual initial orientation of P.

We dedicate the next three subsections to the discussion of the push function for P in its reference orientation. As P aligns with the device, we identify two subsequent stages; namely alignment with the primary plane, and alignment with the secondary plane.

Since we assume that we apply the secondary plane only after the part has aligned with the primary pushing plane, we shall separately discuss the rotational behavior of the part during the two stages. In the next two subsections we discuss the first stage of alignment. The last subsection is devoted to the second stage of alignment.

Alignment with the primary plane

The part P will start to rotate when pushed unless the normal to the primary plane at the point of contact passes through the center-of-mass of P [66]. We refer to the corresponding direction of the contact normal as an *equilibrium* contact direction or orientation.

The contact direction of a supporting plane of P is uniquely defined as the direction of the normal of the plane pointing into P. We study the *radius function* of the part, in order to explain the alignment of P with the primary plane. The radius function $r : ([0, 2\pi) \times [-\pi/2, \pi/2]) \rightarrow \mathbb{R}^+$ maps a direction (ϕ, ψ) onto the distance from c to the supporting plane of P with contact direction (ϕ, ψ).

We first study the planar radius function for a planar part P_p with center-of-mass c_p. The planar radius function easily generalizes to the radius function for a three-dimensional part. According to Definition 2.1, the planar radius function $r_p : [0, 2\pi) \rightarrow \mathbb{R}^+$ maps a direction θ onto the distance from c to the supporting line of P_p with contact direction θ, see Figure 5.4(a). With the aid of elementary trigonometry, we derive that the distance of c to the supporting line of P_p in contact with a fixed vertex v for contact direction θ equals the distance of c to the intersection of the ray emanating from c in direction θ and the boundary of the disc with diagonal (c, v). Combining the discs for all vertices of P_p gives a geometric method to derive r_p. The radius $r_p(\theta)$ is the distance of c to an intersection of the ray emanating from c in direction θ and the boundary of a disc through a vertex of P_p. If there are multiple discs intersecting the ray, $r_p(\theta)$ equals the maximum of all distances from c to the intersection with any disc—a smaller value would not define the distance of a supporting line of P,

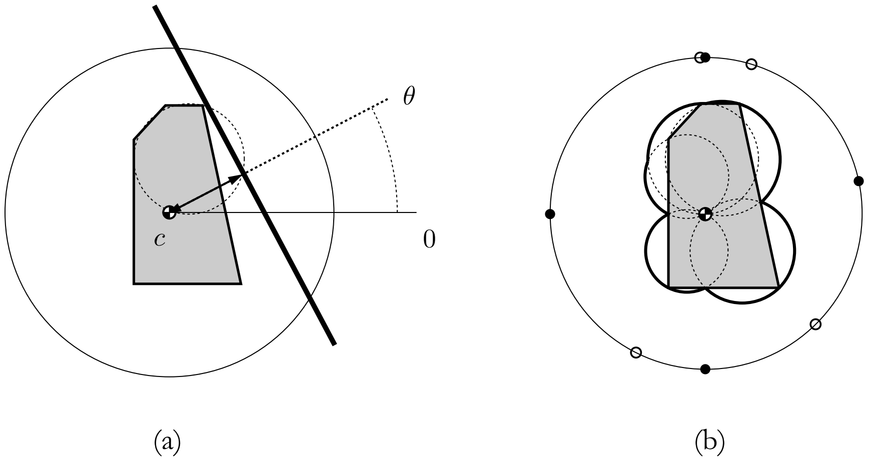


Figure 5.4 (a) The radius for contact direction θ. (b) The bold curves show the radius function for a planar part P. The dots depict local minima, the circles local maxima of the radius function.

but rather line intersecting P. In conclusion, $r_p(\theta)$ equals the distance from c to the intersection of the boundary of union of discs for each vertex of P with the ray emanating from c in direction θ. In Figure 5.4(b), we show a planar part with for each vertex v, the disc with diagonal (c, v). The boundary of the discs is drawn in bold.

The three-dimensional generalization of a disc with diagonal (c, v) is a ball with diagonal (c, v). The three-dimensional radius function $r(\phi, \psi)$ is the distance of c to the intersection of the ray emanating from c in direction (ϕ, ψ) with the union of the set of balls for each vertex of P. We call the boundary of the union the *radius terrain*; it links every contact direction of the primary plane to a unique distance to c.

The radius terrain contains maxima, minima, and saddle points. If the contact direction of the primary plane corresponds to a local extremum, or saddle point of the radius function, the part is at an equilibrium orientation, and the contact direction of the primary plane remains unchanged. If, on the other hand, the radius function of the part for a contact direction of the primary plane is not a local extremum, or saddle point, the gravitational force will move the center-of-mass closer to the primary plane, and the contact direction will change. We assume that, in this case, the contact direction traces a path of steepest descent in the radius terrain until it reaches an equilibrium contact direction. In general, the part can pivot along different features of the part, as the contact direction follows the path of steepest descent towards an equilibrium.

Different types of contact of the primary plane correspond to different features of the radius terrain.

- The contact directions of the primary plane with a vertex of P define a (spherical) patch in the terrain.
- The contact directions of the primary plane with an edge of P define an arc.
- The contact direction of the primary plane with a face of P defines a vertex.

In Figure 5.5, we show different types of contacts of P with the primary plane. Figure 5.5(a) shows an equilibrium contact direction with the primary plane in contact with vertex v_1 of P. The contact direction corresponds to a maximum in the radius terrain. Figure 5.5(b) shows a vertex contact which is not an equilibrium. Figure 5.5(c) shows an equilibrium contact direction for edge (v_3, v_4) of P. Figure 5.5(d) shows a non-equilibrium contact for edge (v_5, v_6). In Figure 5.5(e) we see a degenerate non-equilibrium contact for edge (v_7, v_8), which actually corresponds to a non-equilibrium vertex contact with the primary plane in contact with vertex v_8. The direction of steepest descent in the radius terrain corresponds to a rotation about v_8. Figure 5.5(f) shows a stable equilibrium face contact. The contact direction corresponds to a local minimum of the radius terrain. In Figure 5.5(g) we see a degenerate face contact which corresponds to an edge contact for edge (v_{18}, v_{19}) of P. Figure 5.5(h) shows a degenerate face contact which corresponds to a vertex contact for vertex v_{15}. The alignment of the part to the primary plane is a concatenation of simple rotations, i.e. a rotation about a single vertex or edge. The path of a simple rotation in the radius terrain is either a great arc on a balls with radial (c, v) for a vertex of P, or a part of a intersection of two balls (c, v_1), (c, v_2) for two vertices which is a part of the boundary of a disc. It is easy to see that the projection of the arcs in the radius terrain of any of the simple rotations project to great arcs on the sphere of directions. Hence, during a simple rotation, the contact direction of the primary plane traces a great arc on the sphere of contact directions. During each single stage of alignment, we assume that there is no (instantaneous) rotation about the roll axis.

Computation of the roll after alignment with the primary plane

The mapping of Section 5.1.2 only tells us which feature of the part will be in contact with the primary plane after rotation. It leaves the change in the part's roll out of consideration. Nevertheless, we need to keep track of the roll as P

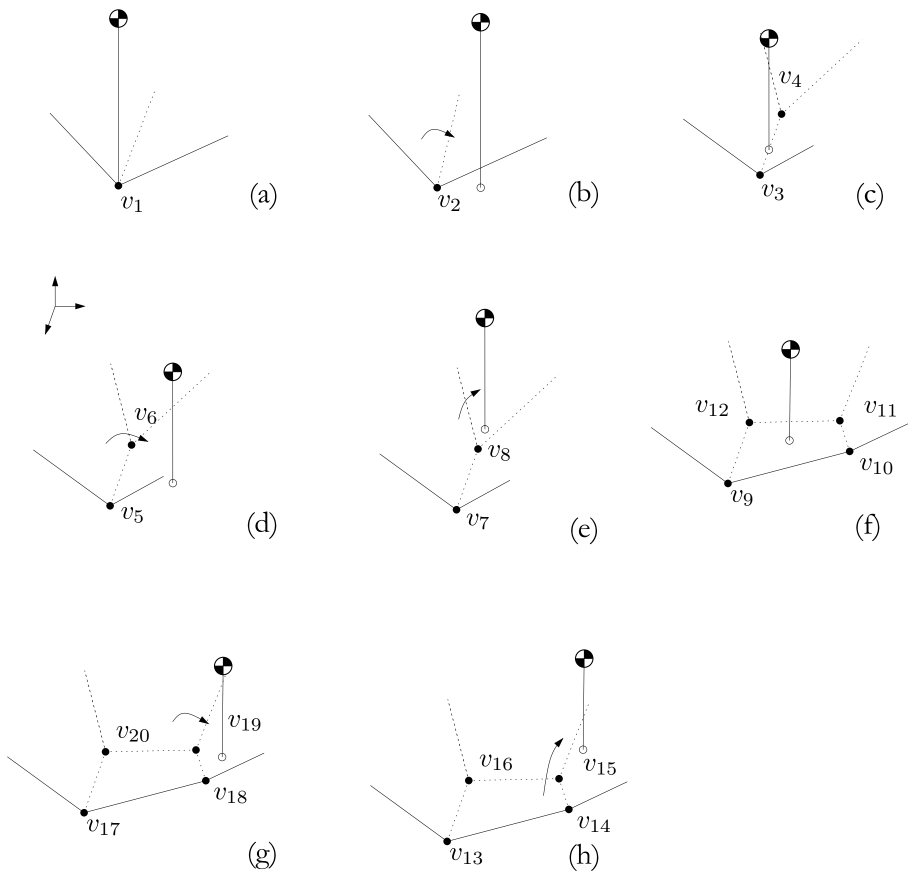


Figure 5.5 Different contacts for the primary plane, with a projection of c onto the primary plane. The primary plane is assumed at the bottom of the pictures.

aligns with the primary plane. We remember that the alignment with the primary plane is a concatenation of simple rotations each corresponding to a great arc on the sphere of contact directions of the primary plane.

With the aid of spherical trigonometry, it is possible to compute the change in roll caused by a reorientation of the primary plane (prior to pushing). Subsequently, we can compute the change in roll for a simple rotation of P. Since the alignment of the part can be regarded as a concatenation of such simple rotations, we obtain the final roll by repeatedly applying the change in the roll of P for each simple rotation in the alignment to the primary plane.

Alignment with the secondary plane

Let us assume that P is in equilibrium contact with the primary plane. The next step in the application of the jaw is a push operation of the secondary (orthogonal) plane. The push action by the secondary plane changes the orientation of the projection of P onto the primary plane. The application of the secondary plane to the part can, therefore, be regarded as a push operation on the two-dimensional orthogonal projection of P onto the primary plane.

The *planar push function* for a planar projection of P $\varpi_{\mathrm{proj}} : [0, 2\pi) \rightarrow [0, 2\pi)$ links every orientation θ to the orientation $\varpi_{\mathrm{proj}}(\theta)$ in which the part P_{proj} settles after being pushed by a jaw with initial contact direction θ (relative to the frame attached to P_{proj}). The rotation of the part due to pushing causes the contact direction of the jaw to change. The final orientation $\varpi_{\mathrm{proj}}(\theta)$ of the part is the contact direction of the jaw after the part has settled. The equilibrium push directions are the fixed points of ϖ_{proj}.

Summarizing, we can compute the orientation of P after application of the jaw. In the next section, we shall show we can always orient P up to symmetry in the push function by means of applications of the jaw.

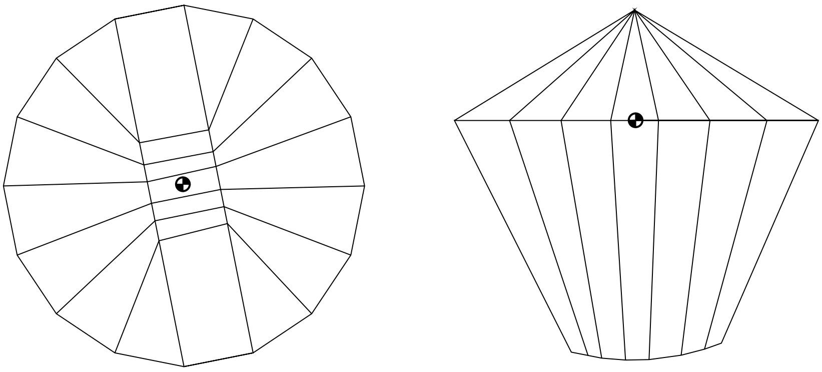
5.2 Orienting a polyhedral part

We will show that any polyhedral part P can be oriented up to periodic symmetry in the push functions of the projections of P onto the primary plane. The part P has at most $O(n)$ equilibria with respect to the primary plane, and any projection of P onto the primary plane has $O(n)$ vertices. Hence, the total number of orientations of P compliant to the jaw is $O(n^2)$. Figure 5.6 shows an example of a part with $\Omega(n^2)$ possible orientations.

Lemma 5.1 *A polyhedral part with n vertices has $O(n^2)$ stable orientations. This bound is tight.*

From the previous section, we know that the pushing jaw rotates P towards one of its equilibrium orientations with respect to the primary plane, and the secondary plane. Let us, for a moment, assume that the contact direction (ϕ, ψ) of the primary plane is known.

We can now redirect and apply the secondary plane. We remember that we assume that applying the secondary plane has no influence on the contact direction of the primary plane. Consequently, the rotations of the part, due to



(a) (b)

Figure 5.6 A part with (n^2) equilibrium orientations. (a) Bottom view. (b) Side view.

applications of the secondary pushing plane, are fully captured by the planar push function of the projection of the part onto the primary plane. Chen and Ierardi [32] have shown that a two-dimensional part with m vertices can be oriented up to (periodic) symmetry by means of planar push plan of length $O(m)$. Consequently, we can orient P in equilibrium contact with the primary plane up to symmetry in the projection of the part onto the primary plane by $O(n)$ applications of the secondary plane.

Lemma 5.2 *Let P be an asymmetric polyhedral part with n vertices. There exists a plan of length $O(n)$ that puts P into a given orientation (ϕ, ψ, θ) from any initial orientation (ϕ, ψ, θ').*

We call the operation which orients P for a single equilibrium contact direction of the primary plane (ϕ, ψ) CollideRollsSequence(ϕ, ψ). We can eliminate the uncertainty in the roll for any equilibrium contact direction of the primary plane. The initialization of the push plan that orients P reduces the number of possible orientations to $O(n)$ by a concatenation of CollideRollsSequence for all equilibrium contact directions of P. Lemma 5.3 will give us a push operation to further reduce the number of possible orientations.

Lemma 5.3 *For every pair of orientations (ϕ, ψ, θ), and (ϕ', ψ', θ') of a polyhedral part there exist two antipodal reorientations of the primary plane which map these orientations onto $(\tilde{\phi}, \tilde{\psi}, \tilde{\theta})$ and $(\tilde{\phi}', \tilde{\psi}', \tilde{\theta}')$, such that $\tilde{\phi} = \tilde{\phi}'$ and $\tilde{\psi} = \tilde{\psi}'$.*

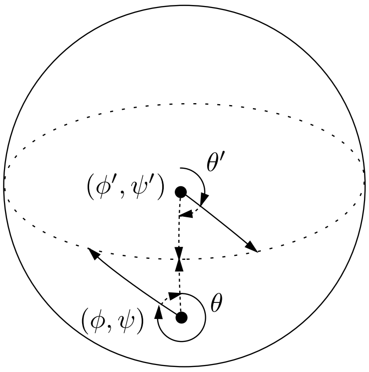


Figure 5.7 Two orientations on the sphere of directions. Their equator is dashed. A desired reorientation of the primary plane is dotted.

Proof: We will prove that there is a reorientation of the primary plane for which the resulting contact directions of the primary plane for P in initial orientation σ and σ' are the same. We focus on the first two parameters of the orientations σ and σ': (ϕ, ψ) and (ϕ', ψ') represent two points on the sphere of directions. We want to find a push direction that maps these two points onto another point (ϕ'', ψ''). See Figure 5.7. Let E denote the great circle consisting of all points equidistant to (ϕ, ψ) and (ϕ', ψ'). E divides the sphere of directions into a hemisphere containing (ϕ, ψ) and a hemisphere containing (ϕ', ψ'). Any reorientation of the primary plane maps (ϕ, ψ) and (ϕ', ψ') onto contact directions which are equidistant to these original contact directions. Let r denote the ray emanating from (ϕ, ψ), in the direction of θ, and r' denote the ray emanating from (ϕ', ψ') in the direction of θ'. Points on the rays (with equal distance δ to the origins) correspond to a reorientation of the primary pushing plane by $(0, \delta)$. Both rays intersect E. We aim for a push direction (ϕ, δ'), with δ' such that the the jaw touches P at an orientation in E. The component ϕ of the push direction changes the direction of the rays emanating from (ϕ, ψ) and (ϕ', ψ') to $\theta + \phi$ and $\theta' + \phi$ resp. We will show that there is ϕ, such that for both orientations the push direction touches the part at the same point. If their first intersection with E is in the same point, we have found a push direction which maps both orientations onto the same face. Since the orientations are in different hemispheres, increasing ϕ will move the intersections of the rays with E in opposite direction along E. This implies that there are two antipodal reorientations of the primary plane where the intersections must pass. These push directions correspond to push directions which map (ϕ, ψ), and (ϕ', ψ') onto the same point. □

We call the basic operation which collides two orientations onto the same equilibrium for the primary plane COLLIDEPRIMARYACTION. Combining Lemma 5.2 and 5.3 leads to a construction of a push plan for a polyhedral part. The following algorithm orients a polyhedral without symmetry in the planar projections of P for equilibrium contact directions of the primary plane.

```
OrientPolyhedron($P$):
▷ After initialization $|\Sigma| = O(n)$
1. while $|\Sigma| > 1$ do
   2.1    pick $(\phi, \psi, \theta)$, $(\phi', \psi', \theta') \in \Sigma$
   2.2    plan ← CollidePrimaryAction($(\phi, \psi, \theta), (\phi', \psi', \theta')$)
          ▷ Lemma 5.3;
          ▷ plan$(\phi, \psi, \theta) = (\phi'', \psi'', \theta'')$, and plan$(\phi', \psi', \theta') = (\phi'', \psi'', \theta''')$
   2.3    for all $(\tilde{\phi}, \tilde{\psi}, \tilde{\theta}) \in \Sigma$
          2.3.1  $(\tilde{\phi}, \tilde{\psi}, \tilde{\theta}) \leftarrow$ plan$(\tilde{\phi}, \tilde{\psi}, \tilde{\theta})$.
   2.4    plan ← CollideRollsSequence($\phi'', \psi''$)
          ▷ Lemma 5.2
   2.5    for all $(\tilde{\phi}, \tilde{\psi}, \tilde{\theta}) \in \Sigma$
          2.5.1  $(\tilde{\phi}, \tilde{\psi}, \tilde{\theta}) \leftarrow$ plan$(\tilde{\phi}, \tilde{\psi}, \tilde{\theta})$.
```

The number of pushes used by this algorithm sums up to $O(n^2)$. Correctness follows directly from Lemmas 5.2 and 5.3.

Theorem 5.4 *Any asymmetric polyhedral part can be oriented by $O(n^2)$ push operations by two orthogonal planes.*

5.3 Computing a push plan

In this section, we present an algorithm for computing a push plan for a three-dimensional part. We know from Section 5.2 that such a plan always exists for asymmetric parts. The push plans of Section 5.2 consist of two stages. During the initialization stage of the algorithm we reduce the number of possible orientations to $O(n)$ different equilibrium contact directions of the primary plane with a unique roll each. The initialization consists of $O(n^2)$ applications of the secondary plane. In the second stage, we run algorithm ORIENTPOLYHEDRON which repeatedly decreases the number of possible orientations of the part by one, by means of a single application of the primary plane followed by $O(n)$ applications of the secondary plane, until one possible orientation remains. Summing up, a push plan of Section 5.2 corresponds to $O(n)$ applications of the primary plane, and $O(n^2)$ applications of the secondary plane.

We maintain the $O(n)$ different orientations which remain after the initialization stage in an array. During the execution of the second stage, we update the entries of the array. Hence, for each application of either of the two planes of the jaw, we compute for $O(n)$ orientations of the array the orientation after application of the jaw.

In order to compute the orientation of P after application of the primary plane, we need to be able to compute the path of steepest descent in the radius terrain. In order to determine the orientation of P after application of the secondary plane, we need to be able to compute the planar projection of P onto the primary plane for stable orientations of P, and we need to compute planar push plans.

We start by discussing the computation of the path of steepest in the radius terrain from the initial contact direction of the primary plane. The path is a concatenation of great arcs on the sphere of contact directions of the primary plane. Lemma 5.5 bounds the complexity of the radius terrain.

Lemma 5.5 *Let P be a convex polyhedral part with n vertices. The complexity of the radius terrain of P is $O(n)$.*

Proof: There exist bijections between the faces of P and the vertices of the radius terrain, the vertices of P and the patches of the radius terrain, and the edges of P and the edges of the radius terrain. Hence, the combinatorial complexity of the radius terrain equals the combinatorial complexity of P, which is $O(n)$. $\square$

In a piecewise-linear terrain with combinatorial complexity n, the complexity of a path of steepest descent can consist of $\Omega(n^2)$ pieces [9]. We shall show, however, that a path of steepest descent in the radius terrain has complexity $O(n)$.

Lemma 5.6 *Let P be convex polyhedral part. A path of steepest descent in the radius terrain of P has combinatorial complexity $O(n)$.*

Proof: A steepest-descent path in the radius terrain consist of simple sub-paths connecting vertices and points on arcs. Thus, the complexity of the path depends on the number of visits of vertices and crossings of arcs. We prove the theorem by showing that the number of visits of a single vertex, and the number of crossings of a single arc is bounded by a constant.

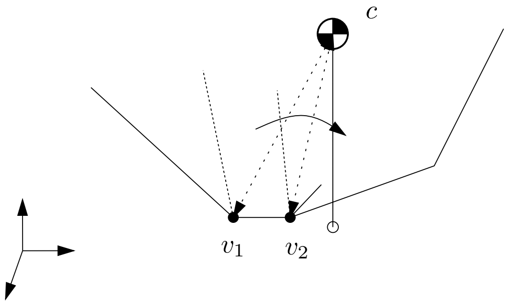


Figure 5.8 The path of steepest descent, crossing an edge of the radius terrain. The distance from v_1 to c is greater than the distance from v_2 to c.

A vertex of the terrain—which corresponds to a face contact—can be visited only once. If the path crosses a vertex, the radius must be strictly decreasing. Hence the path will never reach the height of the vertex again.

We shall show that the path crosses an arc—which corresponds to an edge contact—of the terrain (from one patch to a neighboring patch) at most once. Let us assume that the part is crossing the arc in the terrain which corresponds to a contact of the primary plane to edge (v_1, v_2) of the part. Let us assume that the path in the terrain first travels through the patch of v_1, and then through the patch of v_2. In this case, the part first rotates about v_1, until the edge (v_1, v_2) reaches the primary plane. Instead of rotating about (v_1, v_2), the part subsequently rotates about v_2—the primary plane immediately breaks contact with v_1. Since we assume that the center-of-mass follows the path of steepest descend in the radius terrain, the primary plane can only break contact with v_1 if the distance of v_1 to c is greater than the distance of v_2 to c. See Figure 5.8. Hence for each arc crossing, the part pivots on a vertex with smaller distance to c, and consequently crosses each arc at most once.

Since the number of arcs and vertices of the radius terrain is bounded by $O(n)$, the proof follows. □

In order to compute the path of steepest descent, we need not compute the radius terrain. It is sufficient to use a decomposition of the sphere of contact directions—of which the cells correspond to primary plane-vertex contacts—together with the position of the corresponding vertices on the sphere. We assume that P is given as a doubly-connected edge list. A doubly connected edge list consists of three arrays which contain the vertices, the edges, and the faces of the part. We refer the reader to [10, 84] for a detailed description, and [5] for a discussion on the implementation of the doubly-connected edge list to represent polyhedra. For our purposes, it suffices to know that the doubly-connected edge list allows us to answer all relevant adjacency queries in constant time.

We compute the decomposition of the sphere of contact directions from the doubly-connected edge list of P. We recall that the cells of the arrangement on the sphere of contact directions correspond to plane-vertex contacts. For contact directions at the boundary of a cells, the primary plane is in contact with at least two vertices, and thus with an edge or face of P.

We use the aforementioned correspondence between the part and the arrangement to efficiently compute the latter from the former. For each edge of the part, we add an edge to the arrangement. The vertices of the edge correspond to the contact directions of the primary plane at the faces of the part neighboring the edge. These contact directions are computed in constant time from the edge and a third vertex on the boundary of the face. The connectivity captured by the representation of the part, easily carries over to the connectivity of the arrangement. Hence, the computation of the doubly-connected edge list representing the arrangement on the sphere of directions can be carried out in $O(n)$ time. With each cell of the arrangement, we store the corresponding vertex of the part. Figure 5.9(a) shows the decomposition of the sphere of contact directions for a cubic part.

In the example, each face, each edge, and each vertex of the cube has an equilibrium contact direction of the primary plane. As a consequence, any contact direction which corresponds to a face contact is an equilibrium contact direction and the pivoting stops after a constant number of steps. In Figure 5.9(b), we show the great arcs on the sphere of directions which correspond to the simple rotations of the alignment of the part to the primary plane. Firstly, the part rotates about vertex v_1, until edge e_1 reaches the primary plane. The part continues to rotate about edge e_1, until it finally reaches face f_1.

In order to determine the orientation for a given initial contact direction, we need to determine the contact vertex. In other words, we need to determine which cell of the arrangement corresponds to the contact direction. It is not hard to see that this can be accomplished in linear time, by walking through the arrangement.

Lemma 5.7 *Let P be a polyhedral part with n vertices in its reference orientation. Let $(\tilde{\phi}, \tilde{\psi})$ be a push direction of the primary plane. We can determine the orientation (ϕ, ψ, θ) of the part after application of the primary plane in $O(n)$ time.*

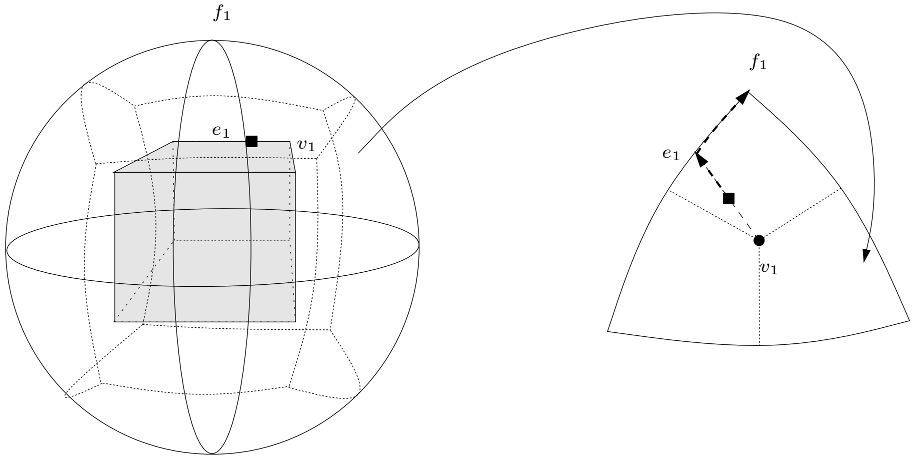


Figure 5.9 (a) The decomposition of the sphere of directions (solid), together with the projection of the part (dotted). (b) The face for which the primary plane is in contact with v_1. The arrows show the contact directions of the primary plane starting at squared the contact point until the part settles on face f_1.

Computing an orthogonal projection of P onto the primary plane can be carried out in linear time per equilibrium by means of an algorithm of Ponce *et al.* [83], which first finds the leftmost vertex of the projection through linear programming, and then traces the boundary of the projection.
The planar push function of a given projection can be computed in $O(n)$ time by checking its vertices. Querying the planar push function can be carried out in $O(\log n)$ time by performing a binary search on the initial orientation.

Lemma 5.8 *Let P be a polyhedral part with n vertices in equilibrium orientation (ϕ, ψ, θ). Let $\tilde{\theta}$ be a push direction of the secondary plane. We can determine the orientation (ϕ, ψ, θ') of the part after application of the secondary plane in $O(\log n)$ time.*

For almost all parts, the computation of a planar push plan of linear length can be done in $O(n)$ time using an algorithm due to Chen and Ierardi [32]. They have shown that there are pathological parts for which they only give an $O(n^2)$ algorithm for computing a push plan. So, the best upper bound on the running time to compute COLLIDEROLLSSEQUENCE for $O(n)$ projections of P is $O(n^3)$. It remains open whether a polyhedral part can have $\Omega(n^2)$ pathological projections, or to improve the bound on the running time in another way. Computing the push direction of the primary plane which maps

two different faces onto the same equilibrium with respect to the primary plane (COLLIDEPRIMARYACTION) can be done in constant time.
Summarizing, the total cost of computing the reorientations of the jaw takes $O(n^2)$ time. The cost of the necessary maintenance of $O(n)$ possible orientations of P is the sum of $O(n^2)$ updates for applications of the secondary plane which take $O(n \log n)$ time each, and $O(n)$ updates for applications of the primary plane, which take $O(n^2)$ maintenance time each. Theorem 5.9 gives the main result of this section.

Theorem 5.9 *A push plan of length $O(n^2)$ for an symmetric polyhedral part with n vertices can be computed in $O(n^3 \log n)$ time.*

5.4 Plates with fences

In this section we will use the results from the preceding sections to determine a design for the feeder consisting of tilted plates with curved tips, each carrying a sequence of fences. The motion of the part effectively turns the role of the plates into the role of the primary pushing plane, and the role of the fences into the role of the secondary pushing plane. We assume that the part quasi-statically aligns to the next plate, similar to the alignment with the primary plane of the generic jaw. Also, we assume that the contact direction of the plate does not change as the fences brush the part, i.e. the part does not tumble over.
The fact that the direction of the push, i.e., the normal at the fence, must have a non-zero component in the direction opposite to the motion of the part, which is pulled down by gravity, imposes a restriction on successive push directions of the secondary plane. Fence design can be regarded as finding a constrained sequence of push directions. The additional constraints make fence design in the plane considerably more difficult than orientation by a pushing jaw.
As the part moves towards the end of a plate, the curved end of the plate causes the feature on which the part rests to align with the vertical axis, while retaining the roll of the part. When the part leaves the plate, the next plate can only push the part from below. This draws restrictions on the possible reorientations of the primary plane, in the model with the generic three-dimensional jaw (see Figure 5.10).
From careful analysis, it follows that the reorientation of the primary plane is within $(-\pi, 0) \times (0, \pi)$ when the last fence of the last plate was a left fence. Similarly, for a last right fence, the reorientation of the primary plane is within $(0, \pi) \times (0, \pi)$.

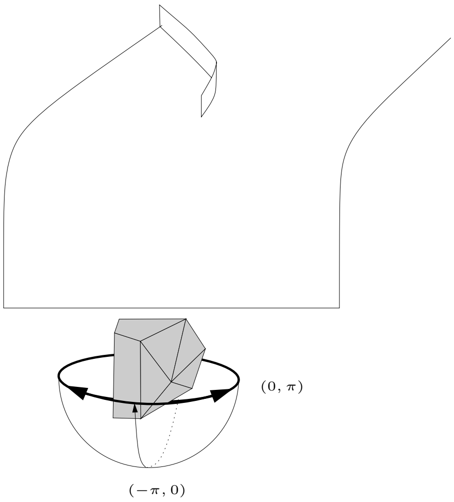


Figure 5.10 The next plate can only touch the lower half of the part.

Theorem 3.6 shows us that it is possible to orient a planar polygonal part (hence a polyhedral part resting on a fixed face) using $O(n^2)$ fences. The optimal fence design can be computed in $O(n^3)$ time.

The gravitational force restricts our possible orientations of the primary plane in the general framework. Fortunately, Lemma 5.3 gives us two antipodal possible reorientations of the primary plane. It is not hard to see that one of these reorientations is in the reachable hemisphere of reorientations of the push direction of the primary plane for two succesive plates.

This implies we can still find a fence and plate design, which consists of $O(n^3)$ push operations.

Theorem 5.10 *An asymmetric polyhedral part can be oriented using $O(n^3)$ fences and plates. We can compute the design in $O(n^4 \log n)$ time.*

5.5 Discussion

We have shown that sensorless orientation of an asymmetric polyhedral part by a sequence of push actions by a jaw consisting of two orthogonal planes is possible. We have shown that the length of the sequence of actions is $O(n^2)$ for polyhedral parts with n vertices, and that such a sequence can be determined in $O(n^3 \log n)$ time.

We have proposed a three-dimensional generalization of conveyor belts with fences [15]. This generalization consists of a sequence of tilted plates with curved tips, each carrying a sequence of fences. A part slides along the fences of a plate to reach the curved tip where it slides off onto the next plate. Under the assumptions that the motion of the part between two plates is quasi-static and that a part does not tumble from one face onto another during its slide along one plate, we can compute a set-up of $O(n^3)$ plates and fences in $O(n^4 \log n)$ time that will orient a given part with n vertices. (As in the two-dimensional instance of fence design, the computation of such a set-up boils down to the computation of a constrained sequence of push actions.)

Our aim in this chapter has been to gain insight into the complexity of sensorless orientation of three-dimensional parts rather than to create a perfect model of the behaviour of pushed (or sliding) and falling parts. Nevertheless, we can relax some of the assumptions in this chapter. First of all, in a practical setting, a part which rests on a vertex or edge rather than on a stable face, will most likely change its contact direction with the primary plane if it is pushed from the side. Hence, we want to restrict ourselves to orientations of P which have stable equilibrium contact directions of the primary plane. After the first application of the jaw, it might be the case that P is in one of its unstable rather than stable equilibria. A sufficiently small reorientation of the jaw in an appropriate direction, followed by a second application of the jaw, will move the part towards a stable orientation though, allowing us to start from stable orientations only.

The computation of the reorientation of the primary plane results in two candidate reorientations. Although extremely unlikely, these reorientations could both correspond to unstable equilibrium contact directions. As mentioned, in a practical situation one wants to avoid such push directions. It is an open question whether there exist parts which can not be oriented without such unstable contact directions.

Our approach works for parts which have asymmetric projections onto the primary plane for all equilibrium contact directions of primary plane. It is an open problem to exactly classify parts that cannot be fed by the jaw.

It is interesting to see how the ideas of this chapter can be extended to other feeders such as the parallel jaw gripper, which first orients a three-dimensional part in the plane, and subsequently drops it onto another orientation. Rao *et al.* [88] show how to compute contact directions for a parallel jaw gripper to move a three-dimensional part from a known orientation to another one. We want to see if this method generalizes to sensorless reorientation.

The algorithm presented in this chapter generates push plans of quadratic, and plate and fence designs of cubic length. It remains to be shown whether this bound is asymptotically tight. Also, it is interesting to find an algorithm which computes the shortest push plan that orients a given part. Such an algorithm would need to decompose the space of possible reorientations of the jaw for P in its reference orientation into regions which map onto different final orientations of P. This requires a proper algebraic formulation of the push function, and a costly computation of the corresponding arrangement in the space of push directions. In contrast to the planar push function, the three-dimensional push function is not a monotonic transfer function. Eppstein [40] showed that, for general part feeders with non-monotonic transfer functions, finding a shortest plan is NP-complete. It is an open question whether we can find an algorithm for computing a shortest plan for the generic jaw that runs in polynomial time.

CHAPTER 6

Traps for vibratory bowl feeders

The oldest and still most common approach to automated feeding is the *vibratory bowl feeder*. It consists of a bowl filled with parts surrounded by a helical metal track [21, 22]. The bowl and track undergo an asymmetric helical vibration that causes parts to move up the track, where they encounter a sequence of mechanical devices such as wiper blades, grooves and traps. Most of these devices are filters that serve to reject (force back to the bottom of the bowl) parts in all orientations except for the desired one. Thus, a stream of oriented

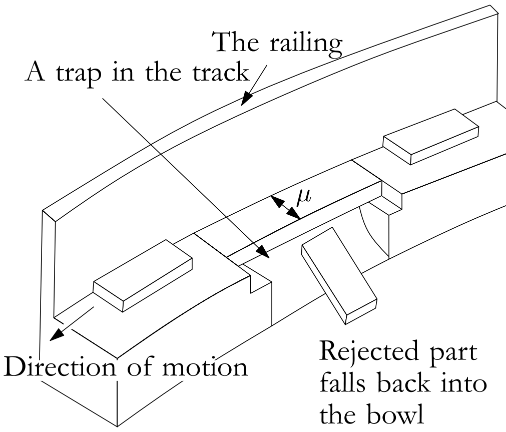


Figure 6.1 Vibratory bowl feeder track [22].

parts emerges at the top after successfully running the gauntlet. In this chapter, we present a framework to filter polygonal parts on a track using traps. A trap is described by removing polygonal sections from the track. A picture of a section of the feeder track is given in Figure 6.1. The parts move from the right to the left on the feeder track. Parts in undesired orientations fall back into the bowl, other orientations remain supported.

Specific to vibratory bowls, researchers have used simulation [11, 53, 69], heuristics [58], and genetic algorithms [35] to design traps. Perhaps closest in spirit to our work is M. Caine's PhD thesis [28] which develops geometric analysis tools to help designers by rendering the configuration-space for a given combination of part, trap, and obstacle. Caine also gives some heuristics to design feeder track features.

This chapter reports on algorithms that design traps with the feeding property. To the extent of our knowledge, no research in systematic algorithmic design of vibratory bowl feeders has been previously conducted.

This chapter is organized as follows. In Section 6.1 we give a geometric model of the bowl feeder; this model is the basis for our algorithms. In Section 6.2 we analyse whether a part in a given orientation will safely move across the trap, or will be filtered out and fall back into the bowl. For a polygonal part with n vertices and a polygonal trap with m vertices, the resulting algorithm runs in $O(n^2 m \log n)$ time. This can be improved to $O((n+m)\log n)$ time if the part and the trap are convex. In Section 6.3 we give algorithms for designing traps in the feeder track. We construct, for example, a gap, which is a rectilinear interruption of the track. Given the geometry of the part, we compute in $O(n^2 \log n)$ time how long the gap should be to establish a feeder. This bound is reduced to $O(n^2)$ for convex parts. We also consider other trap shapes: the balcony, the canyon, the slot, and conclude with a general approach for designing arbitrary polygonal traps. Several algorithms of this chapter have been implemented, and the resulting traps have been succesfully tested in an experimental setup, as will be shown in Section 6.4 . We conclude this chapter in Section 6.5.

6.1 Geometric Modeling

In this section we discuss the geometric properties of the bowl feeder. We address the problem in the plane. Throughout this chapter, P denotes a 2-dimensional polygonal part. The 2-dimensional polygonal trap in the track is denoted by T. The number of vertices of P is denoted by n. The number of vertices of T is denoted by m. The part has a center-of-mass c, which lies inside the convex hull of the part. At c, a fixed coordinate frame is attached, which identifies the zero orientation of the part. The track is slightly tilted towards the railing so the part remains in contact with the railing as it moves along the railing. The radius function, which is defined in Definition 2.1, for the part characterizes the stable orientations of the part against the railing [45]. Each stable orientation of P corresponds to a local minimum in the radius function. The stable orientations of a part can easily be computed in linear time from the description of the part [66]. The orientation of a part is identified by the angle between the reference frame and the y-axis. In our model, the possible orientation of the part is restricted to $O(n)$ different, stable orientations.

In reality, the part is mobile, and slides across a stationary trap in the positive x-direction. It is, however, easier to describe the solutions by viewing the part as stationary, and slide the trap underneath the part (which is obviously equivalent). We assume that the railing of the track is aligned with the x-axis. Throughout the motion of the trap, c is on the y-axis at a fixed distance c_y from the railing, and the part's orientation does not change.

All figures in this chapterhave the railing is coincident with the horizontal axis, and the trap is supposed to move in the negative x-direction. The railing is depicted at the bottom of the figures (see e.g. Figure 6.2).

A placement of the trap, i.e., its horizontal displacement, is denoted by a single value, q. We denote the set of points of the plane covered by trap T at placement q by $T(q)$. The supported area of the part above a trap at placement q is $S(q) = P - \text{int}(T(q))$.

We define how to decide whether a part above a trap in a given placement will fall into the bowl or remain safely on the track. The following definition states that a part is safe if there are three points in P surrounding the center-of-mass that are supported.

Definition 6.1 *Let P be a part with center-of-mass c. Let T be a trap. The part P is safe above the trap at placement q if and only if there exists a triangle Δ_{t_1,t_2,t_3} with $c \in \Delta_{t_1,t_2,t_3}$, and $t_1, t_2, t_3 \in S(q)$. Otherwise, the part is unsafe.*

The following lemmas give us easy ways to decide whether the part is safe or not.

Lemma 6.2 *P above $T(q)$ is safe if and only if $c \in \mathcal{CH}(S(q))$.*

Proof: ($\Rightarrow$) Let P be safe. There is a triangle Δ_{t_1,t_2,t_3}, with $t_1, t_2, t_3 \in S(q)$, and $c \in \Delta$. Clearly, $t_1, t_2, t_3 \in \mathcal{CH}(S(q))$, and, consequently, we have $c \in \mathcal{CH}(S(q))$.
($\Leftarrow$) $c \in \mathcal{CH}(S(q))$. We construct a triangle by computing a triangulation of

$\mathcal{CH}(S(q))$. Clearly there is one triangle in the triangulation which contains c. Furthermore, the vertices of $\mathcal{CH}(S(q))$ are in $S(q)$. □

Lemma 6.3 *P above $T(q)$ is safe if and only if there is no line ℓ through c with $\mathcal{CH}(S(q))$ in the open half plane defined by ℓ.*

Proof: Follows immediately from the previous lemma, because $c \in \mathcal{CH}(S(q))$. □

A *critical placement* of the trap is a placement where c lies on the boundary of $\mathcal{CH}(S(q))$. It follows from Lemma 6.3 that a critical placement can also be characterized by a line through c which touches the boundary of $\mathcal{CH}(S(q))$, that bounds a half place containing $S(q)$. The following lemma gives a third way to characterize a critical placement.

Lemma 6.4 *Let P be a part above a trap $T(q)$. Let c be not in the interior of $\mathcal{CH}(S(q))$. Let ρ_l and ρ_r be two rays emanating from c. Let the left side of ρ_l be tangent to $\mathcal{CH}(S(q))$ and let the right side of ρ_r be tangent to $\mathcal{CH}(S(q))$. The placement of T is critical if and only if the angle between ρ_l and ρ_r is π, or c is a vertex of $\mathcal{CH}(S(q))$.*

Since c lies in the interior of P, we note that if T is convex, c never is a vertex of $\mathcal{CH}(S(q))$. Figure 6.2 depicts a safe part and a trap together with the convex hull of the supported surface. The notion of safeness gives us a tool to formalize whether a part in a given orientation will survive a given trap.

Definition 6.5 *Let P be a part with center-of-mass c in a given orientation. Let T be a trap. The part P is fed if for all placements q, P is save above $T(q)$. Otherwise, P is rejected.*

A trap T has a *critical shape* for orientation σ, if T feeds P in orientation σ, and T has at least one critical placement.
The ultimate goal is to find a trap which will feed only one of the possible orientations of P. A trap with this property is said to have the *feeding property* [2].

6.2 Analyzing a trap

In this section, we analyse the safeness of a part above a given trap. In the first subsection, we discuss the general case of a polygonal part above a moving

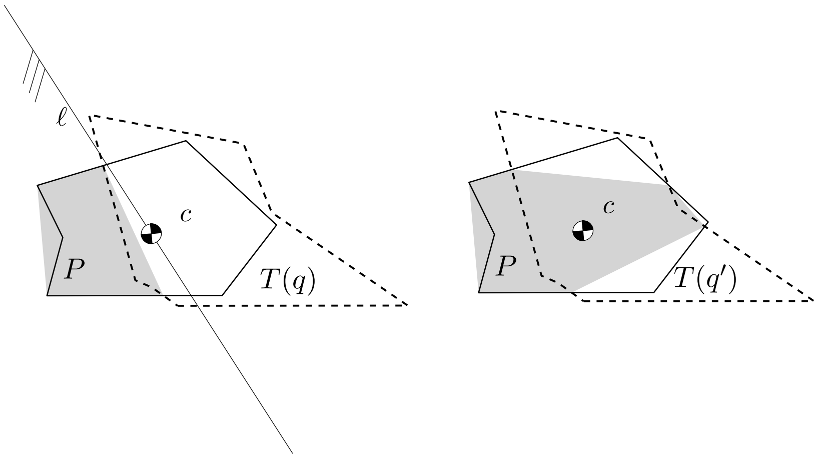


Figure 6.2 The part P and its center-of-mass c above a trap T at different placements q and q'. Placement q corresponds to a unsafe placement. Placement q' is safe. $\mathcal{CH}(S(q))$ and $\mathcal{CH}(S(q'))$ are shaded. The half plane, bounded by line ℓ through c, which contains $\mathcal{CH}(S(q))$ is depicted as well.

polygonal trap. In the second subsection, we give an algorithm with an improved running time, that only deals with convex parts and traps.

6.2.1 Arbitrary polygonal traps and parts

In this section we discuss how to test an orientation of a polygonal part against a polygonal trap in the track. From Section 6.1 we know that there are at most $O(n)$ different possible orientations for the part on the track. We consider one of these $O(n)$ stable orientations of the part. We answer the question whether P in this specific orientation is fed or rejected. We first give a general algorithm which solves the problem in $O(n^2 m \log n)$ time. Then, we give an improved algorithm which works for convex parts and convex traps, and solves the problem in $O((n+m)\log n)$ time.

To determine whether a part will survive a given trap, we sweep the trap underneath the part and check if safeness is retained during the sweep. Lemma 6.3 gives us the idea of the algorithm. Namely, we check whether at any moment during the sweep all points of the convex hull of the supported area are in an open half plane through c. If this is not the case, the part is safe.

We distinguish three types of vertices in the arrangement of the two possibly intersecting polygons T and P: the vertices of P, the vertices of T and the vertices due to the intersections of an edge of P and an edge of T.

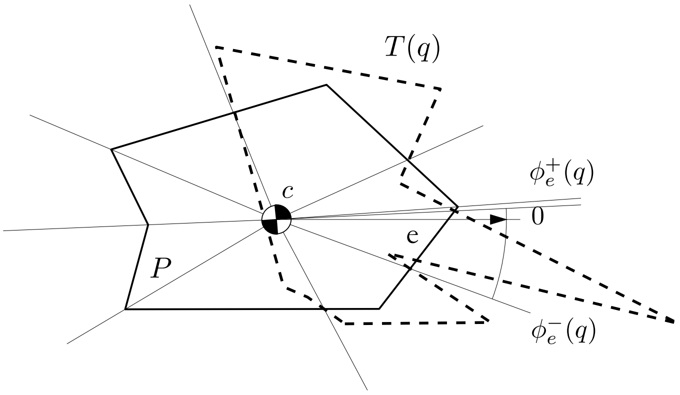


Figure 6.3 The part P and its center-of-mass c above a trap T at placement q. The extremal rays are drawn for all edges of P. For edge e, $\phi_e^-(q)$ and $\phi_e^+(q)$ are shown.

The convex hull $\mathcal{CH}(S(q))$ is equal to the convex hull of a subset of these vertices. Recall that $S(q) = P - \text{int}(T(q))$. The vertices of P can only contribute to $\mathcal{CH}(S(q))$ if they are not in T. The trap T does not contribute to $\mathcal{CH}(S(q))$, but we have to take into account the intersection points of edges of P and T. In general, it is not necessary to take into account every intersection between edges of T and edges of P. It is sufficient, by definition of the convex hull, to only use the (at most) two outermost intersections of each edge of the part.

We compute, for each edge e of the part, the angles of a rays emanating from c through the edge's left- and rightmost point of support during the sweep—for vertical edges, we compute these angles for the lowest and highest point of support. We call these rays *extremal rays*. We are interested whether there is a half-plane bounded by a line through c which contains all the extremal rays. In other words, whether at any time during the sweep, there is a single angular interval greater than π, containing no rays. In the following, we first analyze the complexity of the motions of the rays during the sweep, and then give an algorithm to answer the question of safeness.

The defining features of T and P of these extremal points change during the sweep of the trap. We define ray-angle functions ϕ_e^- and ϕ_e^+. These functions give a mapping from the amount of shift of the trap, to the angles of extremal rays, see Figure 6.3. An edge e need not be supported at all times during the sweep. Hence, these functions are only partially defined. However, since an edge of P intersects with at most $O(m)$ different features of T, each leading to at most a constant complexity curved part of one of the functions, the total combinatorial complexity of ϕ_e^- and ϕ_e^+ is $O(m)$.

Now consider a graph of all ray-angle functions of all edges of P in the (x, ϕ)-plane. Here, an x-value corresponds to the amount of shift of the trap. A vertical line in the graph, i.e. a line with a fixed x-value intersects $O(n)$ functions. If the distance between two function values is greater than π for some x, then the part is unsafe at the corresponding position of the trap, and hence rejected. We check this condition using a frequently used geometric technique called a *sweep line algorithm*. We sweep a vertical line ℓ across the graph in the (x, ϕ)-plane. While we do so, we keep track of the ray-angle functions intersecting ℓ. The description of the ray-angle functions intersecting the sweep line is called the *status* of the sweep line. For details on sweep line algorithms, we refer to the book of De Berg *et al.* [10].

During the sweep, the status needs to be updated at specific values of x. The values at which we update the status of ℓ are called *events*. First, there are events for x-values at which there is an endpoint of a segment of a ray-angle function, since at these values a ray-angle function must be inserted into or removed from the status structure. Also, there are events when two ray-angle functions change order. Finally, there are events at which two neighboring ray-angle functions have distance π.

When we process an event, we first check which type of event we are dealing with. If the event is due to two neighboring ray-angle functions becoming π apart, we check whether the two functions are indeed still neighboring. In this case, the part is unsafe, and we reject it. An event due to a begin- or endpoint of a ray-angle function segment forces insertion or deletion of a ray-angle function in the status structure. The last kind of event raises the need to update the order of the values of the ray-angle functions in the status structure. From the changes in the status structure, we compute new events.

The reader might have noticed that it can occur that, due to an update of the status structure, the events which correspond to two rays which make an angle of π become invalid. We do not remove them from the set of upcoming events, but we recheck, as mentioned before, the validity of these events at the moment they are processed.

In our case, the status structure is implemented as balanced binary tree storing the order in which the ray-angle functions are intersected by the sweep line. Since there are $O(n)$ ray-angle functions present in the intersection with the sweep line, the updates and checks take $O(\log n)$ time. The events are stored in a priority queue. For adjacent functions, we compute their intersections, and the x-value for which they are π apart, and enqueue these events.

There are $O(n)$ partially defined ray-angle functions of combinatorial complexity $O(m)$. Each pair of ray-angle functions intersect at most $O(m)$ times. Thus each pair of ray-angle functions introduces $O(m)$ events. There are $O(n^2)$ pairs of ray-angle functions. Hence, the total number of events is bounded by $O(n^2m)$.

Theorem 6.6 *Let P be a polygonal part with n vertices, and T be a polygonal trap with m vertices. We can report whether P is rejected or fed in $O(n^2 m \log n)$ time.*

6.2.2 Convex traps and parts

In the case of a convex part and a convex trap the problem can be solved more efficiently. In this section we give three lemmas which result in an $O((n+m)\log n)$ algorithm for this case. First, it is shown that the vertices resulting from the intersecting edges of the part and the trap are sufficient to compute safeness of the part. Lemma 6.7 shows that we no longer need to consider the supported area of the part outside the trap, but we can confine with $S(q) \cap T(q)$. Secondly, Lemma 6.8 and 6.9 show that there are only few events, and moreover these events can be processed efficiently, leading to the faster algorithm.

Lemma 6.7 *Let P be a convex part with center-of-mass c at the origin, and $T(q)$ be a convex trap at placement q. P is safe if and only if c is in the convex hull of the vertices of $S(q) \cap T(q)$ or $c \notin T(q)$.*

Proof: ($\Rightarrow$) Trivial.
($\Leftarrow$) We elaborate on the case that $c \in \mathcal{CH}(S(q)) \cap T(q)$, because if $c \notin T(q)$ the part is evidently safe. We will show that $\mathcal{CH}(S(q) \cap T(q)) = \mathcal{CH}(S(q)) \cap T(q)$. We prove this by contradiction. Let us assume that there is a point p in $\mathcal{CH}(S(q)) \cap T(q)$ which is not in $\mathcal{CH}(S(q) \cap T(q))$. This point is in $\mathcal{CH}(S(q))$. Consequently, there are two points p' and p'' in $S(q)$, such that p lies on the edge p', p''. Furthermore, p' and p'' have to lie outside $\mathcal{CH}(S(q)) \cap T$, since otherwise p is in $\mathcal{CH}(S(q)) \cap T(q)$ also. But, since P is convex, any point on this edge is contained in P, and the intersection points of p', p'' with the boundary of $\mathcal{CH}(S(q)) \cap T(q)$ are evidence for p to lie inside $\mathcal{CH}(S(q)) \cap T(q)$. □

We restrict ourselves to the part of the motion when $c \in T(q)$, which is a necessary condition for unsafeness of the part. We maintain rays emanating from c, intersecting the vertices of $\mathcal{CH}(S(q) \cap T(q))$, and check whether the angular distance between any pair of neigboring rays remains smaller than π. We shall not explicitly construct the graph of ray-angle functions, but rather maintain the ordered set of vertices which are intersected by the rays.

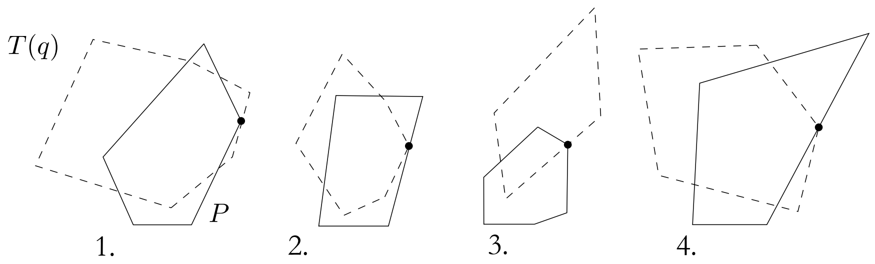


Figure 6.4 The four possible events types.

We need events for each q at which the ordered set of vertices of $\mathcal{CH}(S(q) \cap T(q))$ changes combinatorially. The set could change due to appearance of disappearance of vertices from $S(q) \cap T(q)$, or when three vertices of $S(q) \cap T(q)$ become collinear. The following lemma tells us that any event will coincide with a edge-vertex crossing of P and $T(q)$.

Lemma 6.8 *The combinatorial structure of $\mathcal{CH}(S(q) \cap T(q))$ only changes when a vertex of $T(q)$ move across an edge of P, or an edge of $T(q)$ moves across a vertex of P.*

Proof: Suppose that the combinatorial description of $\mathcal{CH}(S(q) \cap T(q))$ changes when there is no vertex-edge crossing. Clearly, no intersection point of P and $T(q)$ appears or disappears. Thus, the combinatorial change is due to the collinearity of the three moving intersection points, v_1, v_2 and v_3. These three points move along three edges of T, e_1, e_2 and e_3. Consequently, there has to be a line intersecting a convex shape through three edges, which is impossible. This completes the proof by contradiction. □

Hence, there are four possible events where we need to update the combinatorial description of $\mathcal{CH}(S(q) \cap T(q))$ (see Figure 6.4).

1. An edge of $T(q)$ moves across a vertex of P, introducing or deleting a vertex of $\mathcal{CH}(S(q) \cap T(q))$.
2. A vertex of $T(q)$ moves across an edge of P, introducing or deleting a vertex of $\mathcal{CH}(S(q) \cap T(q))$.
3. An edge of $T(q)$ moves across a vertex of P, changing the defining edges of a vertex.
4. An edge of $T(q)$ moves across an edge of P, changing the defining edges of a vertex.

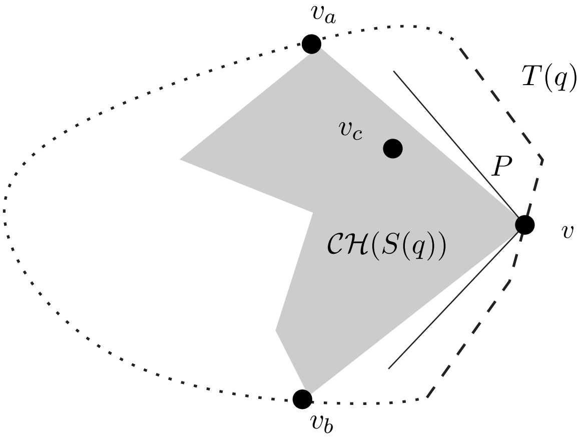


Figure 6.5 An illustration of Case 1 of the proof of Lemma 6.9.

Lemma 6.9 *The events require each a constant complexity update of* $\mathcal{CH}(S(q) \cap T(q))$.

Proof: The convex hull before the event is denoted by $\mathcal{CH}(S(q) \cap T(q))$, and after the event by $\mathcal{CH}(S(q') \cap T(q'))$.

Case 1. Let v denote the vertex appearing or disappearing from the boundary of $\mathcal{CH}(S(q) \cap T(q))$. Let us assume that v appears on the boundary of the convex hull. The other case is similar. We show that $\mathcal{CH}(S(q) \cap T(q))$ changes locally, i.e. an edge (v_1, v_2) changes into two edges (v_1, v) and (v, v_2). Suppose on the contrary, that some of the vertices from the boundary of $\mathcal{CH}(S(q) \cap T(q))$ do not appear on the boundary of $\mathcal{CH}(S(q') \cap T(q'))$, because they are covered by v (Figure 6.5). Let v_a, v, v_b be the neighboring vertices on $\mathcal{CH}(S(q') \cap T(q'))$, after insertion of v. Vertices covered by v are vertices inside the triangle v_a, v, v_b. Let v_c be a covered vertex. Recall that the vertices of $\mathcal{CH}(S(q) \cap T(q))$, as well as the vertices of $\mathcal{CH}(S(q') \cap T(q'))$ are on the boundary of $T(q)$, resp $T(q')$, so v_a, v_b, v_c, and v, are on the boundary of a convex polygon. But, v_c lies in the interior of the triangle v_a, v, v_b, therefore v_c cannot exist and it follows that the transition from $\mathcal{CH}(S(q) \cap T(q))$ to $\mathcal{CH}(S(q') \cap T(q'))$ is indeed local.

Case 2. A vertex of $\mathcal{CH}(S(q) \cap T(q))$ is split into two vertices, or two vertices merge. This is a local change.

Case 3 and 4. The edges defining a vertex change. The direction of the motion of the vertex changes, but the trajectory remains continuous. This is a local change as well. $\square$

By appropriately storing the ordered set of vertices of the convex hull, we can locate the place where the update is necessary in logarithmic time. Hence, the events can be handled in logarithmic time. After preprocessing, we know at which placements of the trap, edges of the trap coincide with vertices of the part and vice versa. Therefore, maintaining the convex hull requires $O((n+m)\log n)$ time.

During the motion of the trap, c always has the same distance to the railing. Therefore, at any moment during the motion, there are only two edges of $\mathcal{CH}(S(q) \cap T(q))$ that c can possibly cross. The intersecting edges of the trap and the part defining these edges might change, though. Every time the description of a relevant edge changes, a new event is generated for the placement at which the center-of-mass will cross the new edge. This is accomplished without increasing the asymptotic running time. From the motion of the center-of-mass, and the motion of the relevant edges we derive placements of the trap at which the center-of-mass leaves the convex hull. We add these placements as extra events. We handle such events as follows. We first check whether the event is still valid, by checking the relevance of the edge associated with the event. If so, we report rejection of the part, otherwise, we discard the event. This gives no extra overhead to the algorithm. The following theorem summarizes the result.

Theorem 6.10 *Let P be a convex polygonal part and let T be a convex polygonal trap. We can report whether P is rejected or fed by T in $O((n+m)\log n)$ time.*

6.3 Design of traps

In this section, we discuss the design of traps. Given a particular part and a collection of traps (e.g. all rectangular traps) the goals is to find a trap in the collection that satisfy the feeder property, i.e. that allow the part to be fed in only one orientation. We start with various collections of rectilinear traps in Sections 6.3.1 through 6.3.4 with increasing numbers of degrees of freedom and conclude with arbitrary polygonal traps in Section 6.3.5.

Figure 6.6 shows a picture of the rectilinear traps we shall present in the next four subsections; balconies, gaps, canyons, and slots. Each with the parameters that define them. The goal of these subsections is to find values for these parameters such that the shape of the resulting trap rejects every orientation of the part, except one.

Clearly, a trap which is entirely contained in another trap will feed all orientations of the latter, and possibly more. For a general pair of traps, on the contrary, neither

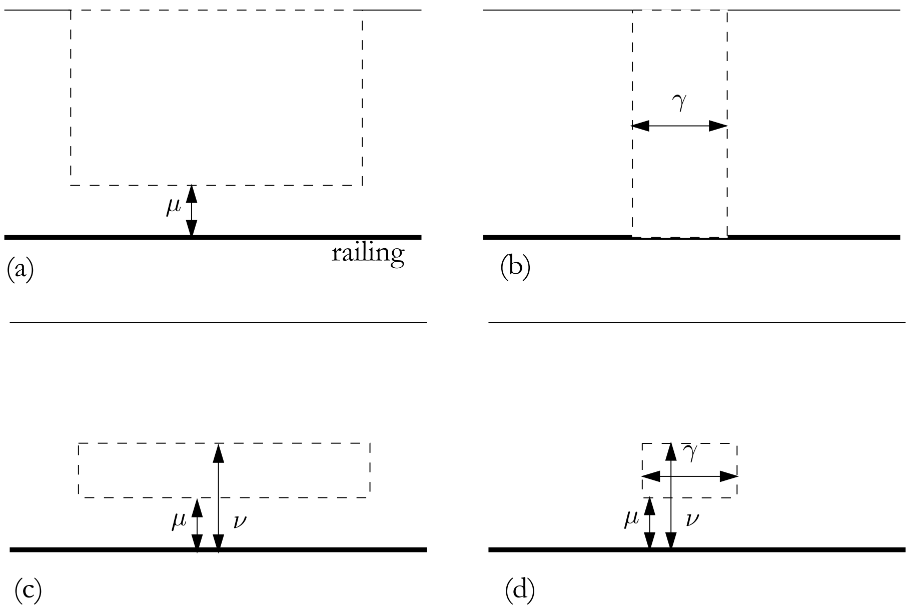


Figure 6.6 The four rectilinear traps of this section: (a) a balcony; (b) a gap; (c) a canyon; and (d) a slot. The thick lines at the bottom of the pictures depict the railing. The line at the top depicts the edge of the track at the inside of the bowl. The traps are dashed.

the first trap need to be contained in the other, nor vice versa. Consequently, it is hard to order different traps based on the rejection or feedability of traps.

We can, however, for a given orientation σ of P, subdivide the parameter space of all possible trap shapes into shapes that feed P in orientation σ, and shapes that reject P in orientation σ.

On the boundaries of the different regions of the subdivision, we find critical trap shapes, which feed the part, but have critical placements—only slightly enlarging such a critical trap shape will turn the critical placement into an unsafe placement, turning the trap into a trap that rejects the part. Combining the subdivisions of the trap shapes for different orientations will, on its turn, lead to trap shapes for which only one orientation is fed.

6.3.1 Balconies

A *balcony* is an interruption of the upper part of the supporting area of the track. The lower boundary, e_l, of this interruption is parallel to the railing. The starting and closing edges of the interruption, e_s, and e_c are orthogonal to the railing. The length of the interruption exceeds the diameter of the part, so that the part can impossibly simultaneously intersect e_s and e_c. A balcony shape is given by the *balcony width*, μ, which is the distance between e_l and the railing (see Figure 6.6(a)).

We assume that the part is in a fixed stable orientation, so one of its convex hull edges is aligned with the railing. We want to identify a critical balcony. If we start decreasing the balcony width μ from the width of the track to zero, then initially the trap will move across the part without causing the part to fall through. At a certain balcony width the part will not survive the balcony, and this clearly remains to be the case for smaller balcony widths. We refer to the smallest balcony width for which the part survives as the *critical balcony width* for this orientation of the part. If the critical balcony width μ of one stable orientation σ is smaller than the critical balcony widths of all other stable orientations then a balcony of width slightly larger than μ (but smaller than all other critical balcony widths) will reject all stable orientations but σ. Hence, this balcony width has the feeding property.

In the following, we show that the critical balcony width for orientation σ corresponds to the distance of the center-of-mass c to the railing. We denote by H, the half plane extending downward from e_l. We observe that $P \cap H \subseteq S(q)$ for any placement q of T. Equality holds for placements for which P is between e_s and e_c. The part in orientation σ is fed by T if and only if P is safe for all placements of T. Since $P \cap H \subseteq S(q)$ for any placement q of T, P is fed if and only if P is safe for placements q of T for which P is between e_s and e_c, and $P \cap H = S(q)$. Consequently, by Lemma 6.2, P is fed if and only if $c \in \mathcal{CH}(P \cap H)$.

A balcony T for which the width μ equals the distance of c to the railing for P in orientation σ has c on the boundary of $\mathcal{CH}(P \cap H)$. Thus, placements q of T for which P is between e_s and e_c have c on the boundary of $\mathcal{CH}(S(q))$ are critical placements, but still feed P in orientation σ. The distance of c to the railing equals the radius of P in direction σ. Therefore, the critical balcony width for P in orientation σ is $\delta(\sigma)$.

Summarizing, there exists a balcony with the feeder property if the open interval between the two smallest radii of all stable orientations of P is non-empty, i.e. there is a unique orientation for which the radius is minimal. Since we can compute all radii of P in linear time [66], we can determine the balcony widths which have the feeder property in linear time as well.

Theorem 6.11 *In $O(n)$ time we can design a balcony with the feeding property for a polygonal part with n vertices, or report that no such balcony exists.*

Proof: The stable orientations and radii corresponding to these orientations of P are computed in linear time. If there is a unique orientation for which the radius is minimal, the feeder is constructed using a balcony slightly higher than this minimum radius. Otherwise, we will always end up with two or more orientations. □

Note that the railing of the track always touches the part at the convex hull. Therefore, the given analysis holds for both convex and non-convex parts. The only parts we cannot feed with a balcony are parts for which the minimal radius is not unique. We might also use a balcony at the other side of the track, facing the railing. This 'reverse' balcony can be used to select part orientations with a radius greater than the width of the reverse balcony. The combination of a balcony and a reverse balcony in succession on the feeder track is very powerful. Actually, we can select any radius ρ, by first rejecting orientations with radii greater than ρ, and then rejecting orientations with radii smaller than ρ. So only parts for which each radius occurs more than once cannot be handled in this way.

6.3.2 Gaps

A *gap* is an interruption of the supporting area that spans the entire width of the track. Both its boundaries are perpendicular to the railing. The shape of a gap can thus be characterized solely by the distance γ between these two parallel boundaries. We shall refer to this distance as the *gap length* (see Figure 6.6(b)).
We again assume that the part is in a fixed stable orientation, so one of its convex hull edges is aligned with the railing. We want to identify a critical gap. If we start increasing the gap length γ from zero to infinity, then initially the trap will move across the part without causing the part to fall through. At a certain gap length the part will not survive the gap, and this clearly remains to be the case for all larger gap lengths. We refer to the largest gap length for which the part survives as the *critical gap length* for this orientation of the part. If the critical gap length γ of one stable orientation σ is larger than the critical gap lengths of all other stable orientations then a gap of length slightly smaller than γ (but larger than all other critical gap lengths) will reject all stable orientations but σ. Hence this gap length has the feeder property. If the largest critical gap length does not correspond to a unique orientation of the part, then there is no gap that can reject all but one orientation of the part, and there exists no gap with the feeding property.
The part is safe if and only if there is a supported triangle around the center-of-mass. This implies that, when the part is unsafe, the supported area of the

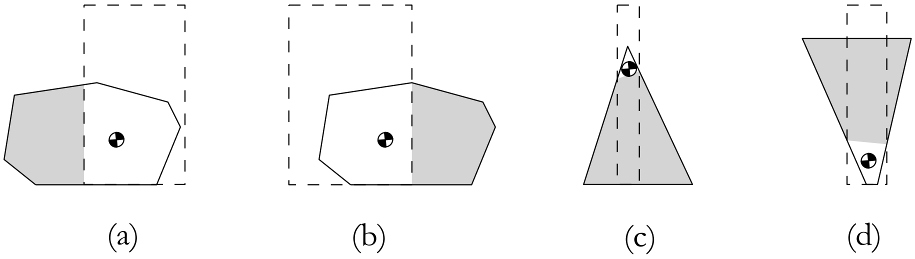


Figure 6.7 The types of unsafe placements. $\mathcal{CH}(S(q))$ is grey. (a) The supports are to the left of the center-of-mass. (b) The supports are to the right of the center-of-mass. (c) The supports are in a half-plane below the center-of-mass. (d) The supports are in a half-plane above the center-of-mass.

part is contained in a half-plane that does not contain the center-of-mass. We distinguish two different types of unsafe, or critical, placements of the part:

1. The supported area of the part is intersected by at most one edge of the gap.
2. The supported area of the part is intersected by both edges of the gap.

In the first type of unsafe placements, the part is only supported to the left (or the right) of the vertical axis through the center-of-mass. For the second type of unsafe placements, the supports are contained in a half-plane below (or above) the center-of-mass. The corresponding critical placements also have supports on a line through the center-of-mass. In Figure 6.7, four types of unsafe placements are given.

The critical gap length for a stable orientation σ equals the smallest gap length associated with the critical placements of the trap of the two types. The first type of placement does not exist as long as the length of the gap does not exceed the radii of the part in the direction $\sigma - \pi/2$ and $\sigma + \pi/2$. Clearly, if one of these radii is less than the gap length, then the part will fall either forward or backward. Thus, the critical gap length is at most $\min\{\text{radius}(\sigma - \pi/2), \text{radius}(\sigma + \pi/2)\}$. It is a bit harder to compute the shortest gap length for which the second type of placement does not occur.

We start by investigating how the supports of the part can be contained in a half-plane below the center-of-mass (the case for the supports above the center-of-mass is similar). Lemma 6.4 is the base for our analysis throughout the rest of this section. We let ρ_l and ρ_r, be two rays emanating from the center-of-mass c, such that $S(q)$ is tangent to the left side of ρ_l, and the right side of ρ_r. Figure 6.8 shows a part at a critical placement of the gap that is supported by two sides of the gap.

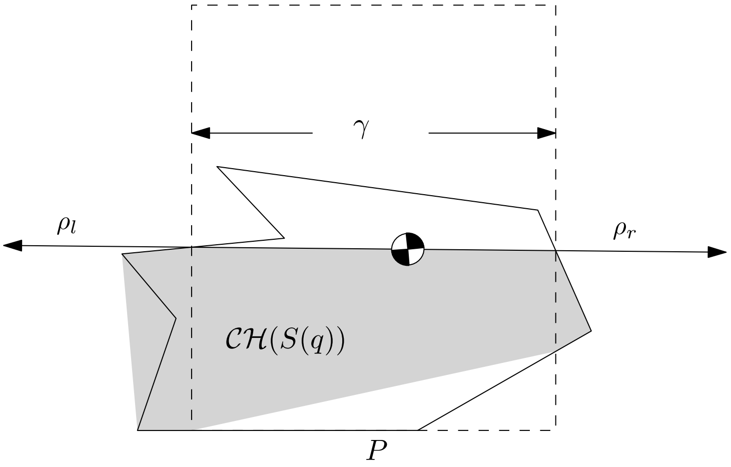


Figure 6.8 A critical placement of the gap for the part. ρ_l and ρ_r are rays emanating from c, touching $\mathcal{CH}(S(q))$ on either side.

The supported area of the part now consists of two regions, one to the left of the gap, and one to the right of the gap. The center-of-mass is in the gap. The widest gap length for which these two regions exist is the gap length which exactly spans the part, supporting the left- and rightmost pieces of the part. Unless c is on the line through the outermost vertices of P, this gap length will not correspond to a critical placement—ρ_l and ρ_r will make an angle unequal π, and the part will be unsupported.

Concludingly, we will have to narrow the gap, until we reach a critical placement, i.e. until the angle between the two rays emanating from c make an angle of π. We define a function ϕ_l, which links γ_l to the angle of ρ_l with the positive vertical axis. The function ϕ_r is defined similarly. Intuitively, moving the left edge of the gap towards c, makes the angle of the left ray with the vertical axis smaller, and moving the rightmost edge of the gap towards c, will make the angle of the right ray with the vertical axis smaller. We search for the combination of both motions, for which the rays emanating from c eventually make an angle of π. In the following, we shall validate the intuition, and show how to compute the different gap lengths which correspond to critical placements.

An important first observation is that ρ_l will always touch $\mathcal{CH}(S(q))$ at the top. Hence, only the points of P with the maximum y-value for each x-value are important. The union of these points is called the *upper envelope* of the part. The upper envelope can be computed in $O(n \log n)$ time, using an algorithm for computing the upper envelope of line segments by Hershberger [51].

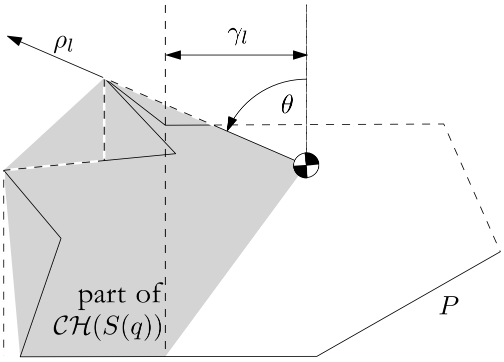


Figure 6.9 The angle of ρ_l with the vertical axis for a given γ_l. The upper tangent envelope of the part is dashed.

The second observation is that supported points of P for γ_l are also supported for γ_l' with $\gamma_l' < \gamma_l$. Hence, the supported area to the left of c only increases as γ_l decreases, and consequently, ϕ_l is monotonic.

From the two observations, it follows that a geometric representation of ϕ_l can be computed in two stages. First, we compute the upper envelope of P. Second, we transform the upper envelope into a shape for which the value of $\rho_l(\gamma_l)$ coincides with the intersection of the vertical line at distance γ_l from the vertical axis. We call this shape the *upper tangent envelope*.

The upper tangent envelope for the left side of P can be incrementally constructed traversing the vertices of the upper hull of P from left to right. We start the traversal at the leftmost vertex v_1 of P. The upper tangent envelope is given by the line segment (v_1, c).

As we travel along a vertex v_i, there are two possibilities to augment the upper tangent envelope. We consider the line segment (v_{i-1}, v_i). If the segment lies (partially) above the upper tangent envelope computed so far, we add two segments to the upper tangent envelope: the segment of (v_{i-1}, v_i) above the upper tangent envelope, and (v_i, c). Otherwise, we discard v_i. We stop at the artificial vertex at the intersection of the vertical axis (through c) and P.

The upper tangent envelope is a representation of ϕ_l and ϕ_r. The value of $\phi_l(\gamma_l)$ is given by the angle of the ray emanating from c to the intersection of the vertical line at distance γ_l from the vertical axis.

The next step is to find all values of γ_l and γ_r for which $|\phi_l(\gamma_l) - \phi_r(\gamma_r)| = \pi$. We start with γ_l at v_1, and find the value of γ_r for which $|\phi_l(\gamma_l) - \phi_r(\gamma_r)| = \pi$. We decrease the value of γ_l, while maintaining $|\phi_l(\gamma_l) - \phi_r(\gamma_r)| = \pi$. From the monotonicity of ϕ_l and ϕ_r it follows that, doing this, γ_r never needs to be decreased. Hence, we can in a single traversal of the edges of the left side of the upper tangent envelope, find corresponding edges of the right side of the tangent envelope for which there are values γ_l and γ_r with critical placements of the trap. Using elementary trigonometry, we compute for each discovered pair of edges the minimal gap length for which there is a critical placement. Altogether, this takes linear time in the complexity of the upper tangent envelope.

Lemma 6.12 *For any orientation of the part, the critical gap length can be computed in* $O(n \log n)$ *time.*

Theorem 6.13 *In* $O(n^2 \log n)$ *time we can design a gap with the feeding property for a polygonal part with* n *vertices, or report that no such gap exists.*

Proof: The stable orientations of P are computed in $O(n)$ time. For each stable orientation we compute the critical gap length. If the minimum of the critical gap length corresponds to a unique orientation, the feeder is constructed using the minimum critical gap length. Otherwise, we will always end up with two or more orientations. □

If the part is convex, the upper tangent envelope of the upper hull of the part is simply the boundary of the part. This allows for a faster computation of the feeder gap length. The critical gap length of a given orientation can now be computed in linear time.

Theorem 6.14 *In* $O(n^2)$ *time we can design a gap with the feeding property for a polygonal part with* n *vertices, or report that no such gap exists.*

Proof: The stable orientations of P are computed in linear time. For each stable orientation we compute the critical gap length. If the minimum if the critical gap lengths is unique, the feeder is constructed using the minimum critical gap length. Otherwise, we will always end up with two or more orientations. □

6.3.3 Canyons

A *canyon* is a rectangular interruption of the supporting area of the track. The lower and upper boundary, e_l and e_u, of this interruption are parallel to the railing. The starting and closing boundary, e_s and e_c, of the interruption are orthogonal to the railing. The length of the interruption exceeds the diameter of the part. Hence, there is no placement q of a canyon for which both e_s and e_c intersect $S(q)$ (see Figure 6.6(c)).

We assume that the part is in a fixed stable orientation, and seek for critical canyons. To this end, we characterize unsafe and critical placements of a canyon. The following lemma allows us to restrict ourselves to placements q of the canyon for which e_s and e_c do not intersect $S(q)$.

Lemma 6.15 *Let P be part. Let T be a canyon. Suppose there is an unsafe placement q of T, with $(e_s \cup e_c) \cap S(q) \neq \varnothing$. There exists an unsafe placement q' of T with $(e_s \cup e_c) \cap S(q') = \varnothing$.*

Proof: We suppose without loss of generality that $e_l \cap S(q) \neq \varnothing$. Let $S_u(q)$ denote the area of $S(q)$ above e_u, $S_l(q)$ denote the area of $S(q)$ below e_l, and $S_c(q)$ the area of $S(q)$ between e_s and e_c. The canyon is longer than the length diameter of the part, so there exists a placement q' for which P lies between e_s and e_c, and clearly $(e_s \cup e_c) \cap S(q') \neq \varnothing$. $S(q') = (S_u(q) \cup S_l(q)) \subset S(q)$. Hence, any triangle that certifies the safeness of P at placement q' of T also exists in $S(q)$, and consequently certifies the safeness at placement q. This implies, by assumption that when P is unsafe at q, P is unsafe at placement q' of T as well. $\square$

A consequence of Lemma 6.15 is that a canyon can be characterized by distances μ and ν of respectively the lower and upper boundary from the railing. Moreover, a critical canyon is characterized by a critical placement q with $(e_s \cup e_c) \cap S(q) = \varnothing$. In the remainder of this section we focus on placements q with $(e_s \cup e_c) \cap S(q) = \varnothing$. We distinguish two types of unsafe, or critical, placements.

1. The supported area of the part is intersected by at most one edge of the canyon.
2. The supported area of the part is intersected by both edges of the canyon.

In the first type of unsafe placements, the part is only supported above the (or below) a horizontal line through the center-of-mass. For the second type of unsafe placements, the supports are contained in a half-plane to the left (or the right) the center-of-mass. The corresponding critical placements also have supports on a line through the center-of-mass.

We denote the height of P in orientation σ by h. Recall that the y-coordinate of c is denoted by c_y. The first type of unsafe placements exists when $\mu = 0$ and

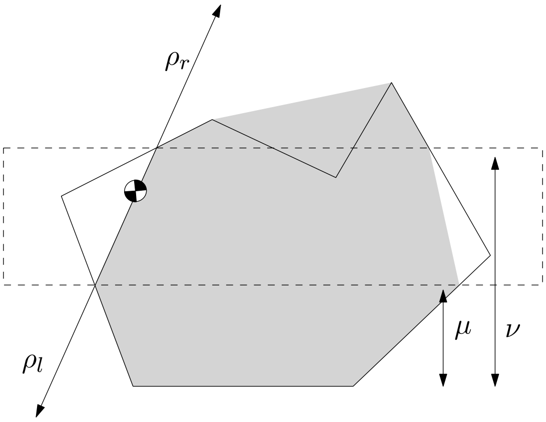


Figure 6.10 A critical placement q for the depicted part. ρ_l and ρ_r are rays emanating from c, touching $\mathcal{CH}(S(q))$ on either side.

$\nu > c_y$, or when $\mu < c_y$ and $\nu > h$. It is a bit harder to compute the second type of critical placements, but they might exist when $0 \leq c_y \leq \nu \leq h$.

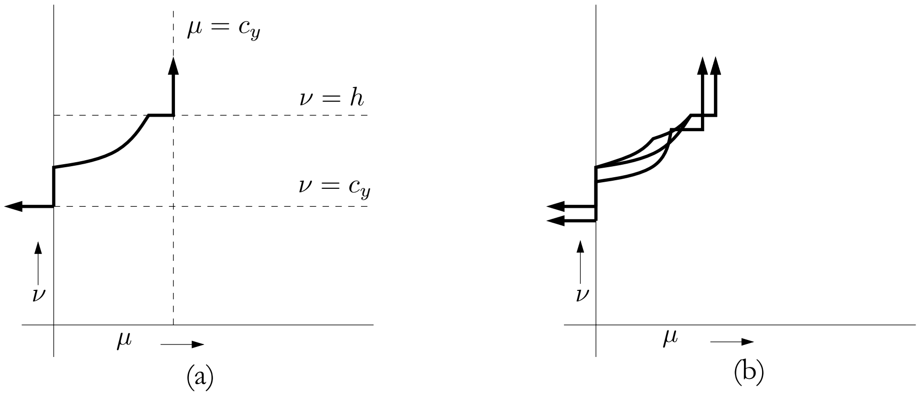
We suppose that $S(q)$ lies in a half plane to the right of c (the other case is similar). We derive the dependency between μ and ν in a way which is rather similar to the discussion in Section 6.3.2.

The supported area of the part, $S(q)$ consists of two regions. One region is above the canyon, and the other is below the canyon. We let ρ_l and ρ_r, be two rays emanating from the c, such that $S(q)$ is tangent to the left side of ρ_l, and the right side of ρ_r. The canyon is at a critical placement if the angle between ρ_l and ρ_r is π. This situation is depicted in Figure 6.10.

We define two functions, ϕ_l and ϕ_r, which link μ to the angle of ρ_l with the horizontal axis, and link ν to angle of ρ_r with the horizontal axis. We make two observations which will shortly lead to a graphical representation of ϕ_l and ϕ_r. Firstly, we observe that ρ_l and ρ_r always touch $\mathcal{CH}(S(q))$ at one of a leftmost point for any y-value. Secondly, we observe that the supported area of the part only increases as we increase μ—causing ρ_l to rotate in clockwise direction—or decrease ν—causing ρ_r to rotate in counterclockwise direction.

Hence, ϕ_l and ϕ_r are monotonic and their representation is given by the left tangent envelope of the part. The left tangent envelope can be computed in $O(n \log n)$ time, similar to the upper tangent envelope of Section 6.3.2.

The next step is to derive the dependency of ν on μ from the left tangent envelope. We start with $\mu = 0$, and compute ν for which $|\phi_l(\mu) - \phi_r(\nu)| = \pi$. Next, we increase μ while maintaining the collinearity of the rays. From the



(a)

(b)

Figure 6.11 (a) The border of a single orientation. (b) The graph of borders of multiple orientations.

monotonicity of ϕ_l and ϕ_r it follows that we never have to decrease ν in this proces. We find a linear number of pairs of edges of the left tangent envelope which are simultaneously intersected by the two rays. For every pair of edges, we can compute the dependency between μ and ν, using elementary trigonometry. We gathered every combination of μ and ν for all critical placements of a canyon for a part in a given orientation. The next step is to combine the combinations for every orientation and select the combinations which only feed one orientation. To find pairs (μ, ν) which satisfy the feeding property, we draw a graph of all critical canyon shapes (μ, ν) for every orientation of P. The graph consists of $O(n)$ curved segments per stable orientation of the part. The segments are connected and the relation between μ and ν is monotonic. We call a connected sequence of these segments a *border*. We draw the border for all possible orientations in the (μ, ν)-plane of all possible canyon shapes.

The border for an orientation σ divides the (μ, ν)-plane into a feeding and a rejecting region for σ. We recall that from the monotonicity of ϕ_l, and ϕ_r it follows that a canyon (μ, ν) that does not have a critical placement for P in orientation σ rejects P in orientation σ if there is a critical placement of a canyon (μ', ν'), with $\mu \geq \mu'$ and $\nu \geq \nu'$. The canyon (μ, ν) feeds P in orientation σ otherwise. Hence, in the graph, the area above the border of σ correspond to canyons that reject P in orientation σ, and the area below the border to canyons that feed P in orientation σ.

Our ultimate goal is to get a pair (μ, ν) that has the feeding property, i.e. the corresponding canyon feeds only one orientation of the part. A valid pair of (μ, ν) must lie above the border of all but one orientations of the part.

The computation of pairs that have the feeding property can be carried out by first finding the uppermost points of all borders in the graph, and then finding the one but uppermost points of all borders. In Figure 6.11 a picture of the border of a single orientation, and the intersecting borders of multiple orientations is depicted. The shape that follows the uppermost points of the graph is called the graph's upper envelope, as the reader might recollect. We compute the upper envelope by means of an algorithm of Hershberger [51] which computes the upper envelope of a set of segments that intersect pairwise at most k times. The combinatorial complexity of the upper envelope is $\Theta(\lambda_{k+2}(n^2))$, where $\lambda_s(n^2)$ is the maximum length of a Davenport-Schinzel sequence of order s on n^2 symbols (see e.g. [95]). The algorithm of Hershberger runs in $O(\lambda_{k+1}(n^2)\log n)$ time. After having computed the upper envelope, we strip the upper envelope from the graph, and run the algorithm of Hershberger again to find the one but uppermost points. In our case, each pair of segments intersect at most twice. Since $\lambda_3(n^2) = n^2\alpha(n)$, where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function, we obtain the following theorem.

Theorem 6.16 *In* $O(n^2\alpha(n)\log n)$ *time we can design a canyon with the feeding property for a polygonal part with* n *vertices, or report that no such canyon exists.*

In the convex case, the running time remains the same, since the final step which computes the upper envelope dominates the running time of our algorithm.

6.3.4 Slots

A *slot* is a rectangular interruption of the supporting area of the track. The lower and upper boundary, e_l and e_u, of this interruption are parallel to the railing. The starting and closing boundary, e_s and e_c, of the interruption are orthogonal to the railing. The distances of the lower and upper boundary from the railing, are specified by μ and ν respectively. The length of the interruption is γ (see Figure 6.6(d)).

The strategy to determine critical placements of the part is a combination of the approach of Section 6.3.2 and 6.3.3. We distinguish two types of unsafe, or critical, placements.

1. The supported area of the part is intersected by at most one edge of the slot.
2. The supported area of the part is intersected by more than one edge of the slot.

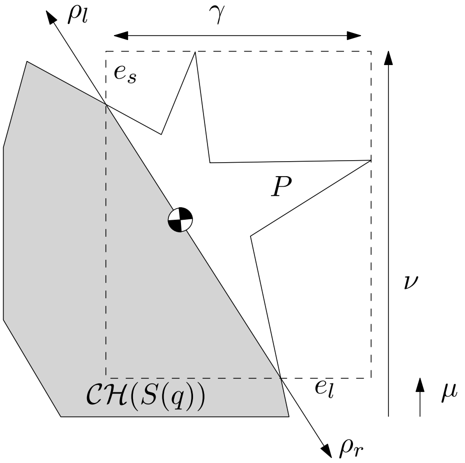


Figure 6.12 A critical slot shape for the depicted part. Parameters γ and μ establish a critical placement certified by the rays. Increasing ν still yields a critical slot shape.

In order to characterize the critical placements of a slot, we recall Lemma 6.4. A slot T is at a critical placement q if there are two rays emanating from c in opposite direction, which are both tangent to $\mathcal{CH}(S(q))$.

Since only two rays determine a critical placement, two edges of the slot are sufficient to determine a critical placement. Consequently, at most two out of three of the parameters which describe a slot are necessary to describe a critical slot. In other words, each critical slot has at least one parameter which is 'free', i.e. there is at least one parameter which can be varied without affecting the criticality of the slot shape. Figure 6.12 shows a critical slot shape with free parameter ν; increasing ν still yields a critical slot shape.

A slot shape is given by the triple (μ, ν, γ). We embed the space of all slot shapes in $\mathbb{R}^3$ and generalize the idea of a 2-dimensional graph of borders in $\mathbb{R}^2$ from Section 6.3.3 to surfaces of critical slot shapes in $\mathbb{R}^3$. This collection of surfaces subdivides the space of all slot shapes into regions of feeding and rejecting slots. The computation of critical slot shapes for P in orientation σ is rather similar to the computation of the critical gap lengths, or canyon shapes of the previous sections. For any pair of boundary edges of the slot, we determine the relation between the angle of collinear rays touching $\mathcal{CH}(S(q))$, and the corresponding slot parameters. This results in a collection of $O(n)$ critical surfaces in the space of slot shapes.

We briefly discuss an algorithm which computes the resulting subdivision of $\mathbb{R}^3$. For each orientation there are $O(n)$ surfaces of constant algebraic complexity.

Thus the arrangement consists of $O(n^2)$ surfaces in a 3-dimensional space. We can compute sample points of the cells in the subdivision, using an algorithm by Basu *et al.* [8]. This algorithm also computes at which side of the surfaces the sample points are located, hence we can determine which orientations are fed, and which orientations are rejected for any sample point. Lemma 6.17 gives the computation time and the number of points computed by their algorithm.

Lemma 6.17 [8] *Let $\mathcal{P} = \{P_1, \ldots, P_\xi\}$ a set of surfaces of constant algebraic degree d in R^l. Then there exists an algorithm that outputs sample points of all semi-algebraically connected components induced by the set $\mathcal{P}$. The complexity of the algorithm is bounded by $\xi^{l+1}d^{O(l)}$.*

In our case $l = 3$ and $\xi = n^2$. Hence, in $O(n^8)$ time we can compute representatives covering all combinatorially different slot shapes. For each representative, we test whether it has the feeding property. If there does not exist such a representative, there is no slot shape that has the feeding property. The following theorem summarizes the result of this section.

Theorem 6.18 *In $O(n^8)$ time we can design a slot with the feeding property for a polygonal part with n vertices, or report that no such slot exists.*

The running time of the algorithm of this section can most likely be improved. The main goal of the presentation of the slot feeder is, however, to give an introduction to computing trap shapes from subdivisions of higher dimensional trap shape spaces. In the next section, we shall compute traps with an arbitrary number of degrees of freedom.

6.3.5 Arbitrary polygonal traps

In this section we will show how to design a general trap. Our goal here is not to provide an optimal algorithm, but to give a general framework.

The proposed general trap is a polygon with k vertices. The position of each vertex of the polygon is specified by two parameters. This implies that a general polygon can be specified by $2k$ parameters.

The problem of this section is as follows. Let P be a polygonal part and let k be an integer. Design a polygonal trap with k vertices such that P is rejected by the trap in all but one stable orientation, as the trap moves across P. Like in the previous sections, we construct a subdivision of the space of possible trap

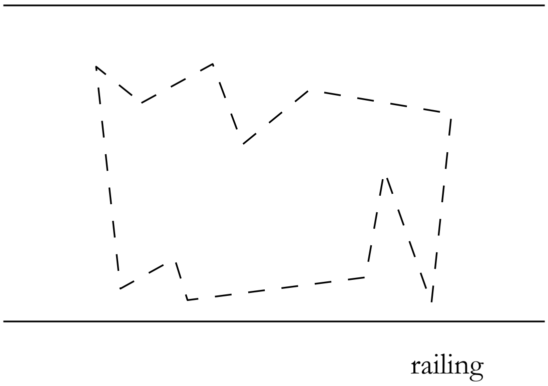


Figure 6.13 An arbitrary polygonal trap. The position of the vertices are parameterized.

shapes. Since a trap shape is determined by $2k$ parameters, the trap shape space in this section is $\mathbb{R}^{2k}$. The computations which lead to a subdivision of the trap shape space will be carried out in a larger dimensional space. More specifically, the computations will be carried out in an Euclidean space which is spanned by the $2k$ parameters of the trap: the position of the part and the (x, y)-plane.

We compute surfaces which correspond to critical placements of the trap. We follow an approach which is related to robot motion planning, using a cell decomposition. We refer the reader to Latombe's book [56] for an overview of robot motion planning, and to the paper of Schwartz and Sharir [94] on a solution to the general motion planning problem, using an arrangement of higher-dimensional, algebraic surfaces.

Our approach to compute the safe placements of the part uses Tarski sets, which are semi-algebraic sets. For ease of presentation, we again describe the problem as if the part will remain stationary during the motion, and the trap moves across the part.

We shall denote the trap, specified by a $2k$-dimensional vector $\tau \in \mathbb{R}^{2k}$, at position $q \in \mathbb{R}$, by $T_\tau(q)$. Let $\Lambda_{\mathrm{int}(T_\tau(q))}(\cdot)$ be the defining formula of the translated trap in $\mathbb{R}^2 \times \mathbb{R}^{2k}$, i.e. $\Lambda_{\mathrm{int}(T_\tau(q))}(v)$, $v \in \mathbb{R}^2$ is true if and only if $v \in \mathrm{int}(T_\tau(q))$. Let $\Lambda_P(\cdot)$ be the defining formula of the part in $\mathbb{R}^2$, i.e. $\Lambda_P(v)$, $v \in \mathbb{R}^2$ is true if and only if $v \in P$. We assume that $\Lambda_P(\cdot)$ and $\Lambda_{\mathrm{int}(T_\tau(q))}(\cdot)$ are semi-algebraic sets. The intersection of the part and the supporting area of the track, $P - \mathrm{int}(T(q))$ is denoted by the following Tarski set $\mathcal{S}_1$.

$$\mathcal{S}_1 = \{(v, \tau, q) \in \mathbb{R}^{2k+3} \mid \Lambda_P(v) \wedge \neg\Lambda_{\mathrm{int}(T_\tau(q))}(v)\}.$$

To determine the safe placements of the part, we take points of S_1, and interpolate between points with the same q and s, i.e. we construct the convex hull of the supported area of the part in the direction of the plane for each trap size and trap position. This leads to the set

$$\begin{aligned}\mathcal{S}_2 = \{(v, \tau, q) \in \mathbb{R}^{2k+3} \mid \exists v' \in \mathbb{R}^2 \exists v'' \in \mathbb{R}^2 \exists i \in [0,1] :\\ v = iv' + (1-i)v'' \wedge\\ (v', \tau, q) \in \mathcal{S}_1 \wedge (v'', \tau, q) \in \mathcal{S}_1\}.\end{aligned}$$

Remember that a placement is safe, if its center-of-mass is inside the convex hull of the supported area of the part. The safe trap shapes in $\mathbb{R}^{2k}$ are the shapes for which there exist no unsafe placement of the trap. A few easy transformations transform the set $\mathcal{S}_2$ into a lower-dimensional arrangement which captures the safeness of the part. The only portion of the arrangement that is of interest is the portion which corresponds to the center-of-mass of P. Therefore, we intersect S_2 with the $(2k+1)$-dimensional space corresponding to the position of the center-of-mass. This results in

$$\mathcal{S}_3 = \{(\tau, q) \in \mathbb{R}^{2k+1} \mid (c, \tau, q) \in \mathcal{S}_2\}.$$

If, during the motion, the center-of-mass is not supported at some placement, the part is rejected. Therefore, we project the complement of the arrangement, onto $\mathbb{R}^{2k}$, obtaining

$$\mathcal{S}_4 = \{\tau \in \mathbb{R}^{2k} \mid \exists s \in [0,1] : (\tau, q) \notin \mathcal{S}_3\}.$$

We now have a description of a subdivision of the trap shape space $\mathbb{R}^{2k}$ for a single stable orientation into fed and rejected cells. For each orientation of the part, we compute this arrangement. The next step is to merge the m different $\mathcal{S}_4$ arrangements, and find a cell for which all but one orientation is rejected. Let us denote $\mathcal{S}_4$ for orientation i by $\mathcal{S}_4(i)$. In the remainder of this section, we discuss how to compute a trap which only feeds the first stable orientation of the part, and reject all other orientations of the part, if such a trap shape exists. Repeating this procedure for the other orientations completes our extensive search for a feeder. Possible trap shapes which feed the first orientation are given by

$$\mathcal{S}_5 = \{\tau \in \mathcal{S}_4(1) \mid \forall o \in [2, \ldots, m] : \tau \notin \mathcal{S}_4(o')\}.$$

Note that o is not a real algebraic variable, and its universal quantifier represents an ordinary for-loop. Unfortunately, the remaining quantifiers found in the expansion of $\mathcal{S}_4(i)$ are harder to deal with.

To be able to eliminate these quantifiers, we first transform $\mathcal{S}_5$ into an equivalent sentence with the quantifiers to the left. This is standard procedure and can be found in e.g. the book of Mishra [71]. We denote the resulting formula by $\mathcal{S}$. Traditionally, elimination of the quantifiers in $\mathcal{S}$ can be done by Collins' decomposition [36] which is a (doubly) exponential algorithm in the number of vertices of the trap. The output of Collins' algorithm are the cells in the arrangement in $\mathbb{R}^{2k}$ of any dimension.

We can improve the running time of the quantifier elimination algorithm, using recent techniques from real algebraic geometry. For a survey, we refer the reader to Heintz *et al.* [50], the book of Mishra [71], and a paper by Chazelle [30]. For a comprehensive introductory discussion to the results cited in the following, we refer to the thesis of Basu [6].

We observe several interesting properties of our formula $\mathcal{S}$. Although the number of free variables of $\mathcal{S}$ is only bounded by $O(k)$—the number of vertices of the trap—the number of quantified variables is bounded by a constant. Also, the degree d of the polynomials in $\mathcal{S}$ is bounded by a constant.

If we would settle for only the semi-algebraic description of the surfaces (without decomposing the space of possible trap shapes into connected components), we could use the recent algorithm of Basu [7], which benefits from the special properties of our set $\mathcal{S}$, and uses singly exponential time in our case. The following lemma states that we can remove the quantifiers from our formula $\mathcal{S}$ by computing a set of surfaces which induces a fine subdivision of the trap shape space which is equivalent to our formula $\mathcal{S}$.

Lemma 6.19 [7] *Let l and ω be constants, and $\mathcal{P} = \{\wp_1, \ldots, \wp_\xi\}$, be as set of ξ polynomials each of constant degree d, in $l + \kappa$ variables, with coefficients in a real closed field R and*

$$\Phi(Y) = (Q_\omega X^{[\omega]}) \cdots (Q_1 X^{[1]}) F(\wp_1, \ldots, \wp_\xi),$$

a first-order formula, where $Q_i \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$, $Y = (Y_1, \ldots, Y_\kappa)$ is a block of κ free variables, $X^{[i]}$ is a block of a constant number of variables, and $F(\wp_1, \ldots \wp_\xi)$is a quantifier-free Boolean formula with atomic predicates of the form $\wp_i(Y, X^{[\omega]}, \ldots, X^{[1]})$ $\{<, >\}0$. Moreover, let every polynomial in $\mathcal{P}$ depend on at most a constant number of the Y_j's.

Then, there exists an equivalent set of surfaces, $\Psi(Y)$ of size $\xi^{O(1)} d^{O(\kappa)} |F|$, where $|F|$ is the length of the formula F. The algebraic degrees of the surfaces in $\Psi(Y)$ are bounded by a constant.

In our case, ξ, and $|F|$ are $O(knm) = O(kn^2)$, and κ is two times the number parameters of the trap. Hence, there is a constant c, such that the output of the algorithm of Basu is a quantifier free formula of size $O((kn)^c d^{O(k)})$ and has constant degree polynomials. The algorithm uses $O((kn)^c d^{O(k)})$ arithmetic operations. The polynomials in the quantifier free formula represent surfaces of constant degree in the trap shape space. The output of the algorithm is a set of surfaces which subdivide the space of possible trap shapes into regions for which the sets of rejected and fed orientations are fixed.

From Lemma 6.17 it follows that we can compute sample points in every connected component of the subdivision. We fill in the variables of Lemma 6.19. From the number of surfaces, it follows $\xi = O((kn)^c d^{O(k)})$. The surfaces are embedded in $\mathbb{R}^{2k}$, hence $l = 2k$. We conclude that we can compute representatives covering all combinatorially different k-vertex trap shapes using $O((nk)^{O(k^2)})$ arithmetic operations. For each representative, we test whether the resulting trap only feeds the first orientation of P. If there does not exist such a representative, there is no k-vertex polygon which only feeds the first orientation of P. Repeating the algorithm for each stable orientation of P yields the following result.

Theorem 6.20 *In $O((nk)^{O(k^2)})$ time we can design a polygonal trap with k vertices with the feeding property for a polygonal part with n vertices, or report that no such trap exists.*

6.4 Experimental results

Some of our algorithms have been implemented and various traps have been shown to work with the aid of a physical inline vibratory feeder. The traps were designed by our algorithms and cut with a milling machine. The resulting feeder successfully feeds a stream of parts. See Figure 6.14 for a picture of the feeder, which was built by Cheung and Smith, U.C. Berkeley, CA, U.S.A. For more information on the experiments, see [12].

6.5 Discussion

In this chapter, we have presented a geometric framework for the trap design problem, and reported algorithms for the analysis and design of various traps for polygonal parts moving across a feeder track. We are not aware of any previous algorithms for the systematic design of vibratory bowl feeder traps.

Some of our algorithms have been implemented and various traps have been shown to work, both in simulation and in practice [12]. Many open problems

In this thesis, we did not try to immediately solve problems as they occur in a practical environment, but rather build a solid foundation for part feeding algorithms. It is, nevertheless, interesting to see how our algorithms perform in experimental setups. We refer the reader to Section 6.4 for a picture of an in-line vibratory feeder with traps that successfully feeds a stream of parts. The feeder was built by Cheung and Smith, see also [12]. In his thesis, Wiegley [110] reports the results of physical experiments with fences across a conveyor belt related to Chapter 3 of this thesis. Also Rusaw *et al.* [93] report on empirical experiments with fences in the presence of friction.

I hope this thesis will motivate researchers to further explore the field of part feeding, and robotic manipulation in general, from an algorithmic perspective. This will hopefully lead to algorithmic solutions that will be applicable to real-world devices.

It might be the case, though, that the wanted pose of the part also starts to drop. This is undesirable. Therefore, we have to adjust the vibration of the bowl in such a way the we can shrink the trap again to a size which still has the property that just one orientation is always safe, and the other orientations are always rejected.

Using parametric searching in geometry [1, 33, 57, 70, 96, 103, 104], we can design an algorithm which computes the maximum possible growth factor of a convex trap, which still assures that the wanted pose of a part is safe. This maximum growth factor translates to the amount of sensitivity to uncertainty in the position of the part for a specific trap. An algorithm based on parametric searching would only lead to a minor increase in the running time of the algorithm which test whether a specific orientation of P is fed or rejected. Although we did not work out the details, we expect to find an $O((n+m)\log^3 n)$ time algorithm to determine the maximum amount of vibration allowed such that P is fed in only the desired orientation

In the future, we want to study the effects of combining multiple traps, and we plan to deal with curved parts. Another interesting direction of research is the extension to three-dimensional parts. Since three-dimensional parts easier get stuck due to their heights, wider trap shapes might be needed to force rejected parts to fall back into the bowl.

CHAPTER 7

Concluding remarks

We have focused on geometric algorithms for part feeding problem, i.e. the problem of how to orient parts of the same geometry and feed them in a unique orientation to a robot or other device. From a practical point of view, part feeding is crucial for the throughput of an assembly line. Unfortunately, practitioners who design and operate part feeders perform their tasks on a trial-and-error basis, which is obviously time consuming and error prone.

Theoretical literature has focused on easy instances of the feeding problem, like the problem of orienting a flat, two-dimensional polygonal, convex, part in a world without friction and dynamics. In this simple abstraction of the real world, some questions have been answered and some algorithms have been developed. The beauty of the simplified world is found in the clear link from part feeding problems to problems in computational geometry. In this thesis we treated part feeding from a computational geometric point of view. Our goal was to answer basic questions which will hopefully later generalize into diverse directions: one can extend results from polygonal to curved parts; from two-dimensional to three-dimensional; from frictionless contacts to contacts with friction; from quasi-static to dynamic motion, etc. Clearly, the solutions obtained are not of direct practical interest. But they can indicate directions for solutions and, in particular, show the potential possibilities and limitations of certain part feeding devices in practice.

The work in this thesis concentrated on four different types of feeders. We looked at rigid fences that brush against a part, thus reorienting it. We have been able to give a positive answer to the previously open question of completeness for fence design, and showed that polygonal parts can be realigned into a unique orientation this way. We have shown that polygonal parts with n vertices in general position can be oriented by a sequence of $O(n)$ fences. For other parts, this bound is $O(n^2)$. Open is the question of extending fence design for curved objects, and a tight bound on the number of fences needed in the design. We have given the first polynomial algorithm for computing fence designs. The algorithm in this thesis improves the previously published result of Berretty *et al.* [15]. Moreover, we have given a bound on the complexity of any part feeding problem with monotonic transfer functions.

Motivated by the technique of inside-out grasping that is used in practice, we considered a pulling finger which orients a planar part with elevated edges by performing a sequence of pull operations. We have shown that convex polygonal parts with n vertices in general position can be oriented by $O(n)$ pull operations. We have proven the existance of non-convex parts that cannot be fed by a sequence of pull operations, showing the limitations of this approach in practice. Open is the question of precisely characterizing the class of unfeedable parts, and giving tight bounds on the length of the pull plan for feedable parts for which the vertices are not in general position.

Most of the efforts in the area of theoretical research on part feeding have focused on feeding two-dimensional parts. We have described the first complete method to feed a three-dimensional polyhedral part by a sequence of push operations. We have shown that any polyhedral part for which the vertices are in general position can be oriented by a sequence of plates and fences. We presented a generic jaw consisting of two orthogonal planes to push three-dimensional parts, and showed that we can orient an n-vertex part using $O(n^2)$ applications of the generic jaw. The algorithm to compute a sequence of push operations runs in $O(n^3 \log n)$ time. Although far from a realistic device, the results show potential use and can form the basis for further research in three-dimensional part feeding.

A part feeder that is widely used in practice is the bowl feeder. The design of bowl feeders is nowadays a practitioners' task. In literature results on simulation, heuristics or genetic algorithms to design traps can be found. Also design-aid and analysis tools have been reported. We gave algorithms to systematically design traps in the track of the bowl feeder for planar parts. We touched questions in the area of uncertainty in the shape or the position of the part. Open is the question of extending these bowl feeder design algorithms to three dimensions, and finding better algorithms in terms of efficiency. In practice, bowl feeders use other gadgets besides traps to orient parts. Systematic design of these features should be studied as well; especially when one wants to design bowl feeders to feed three-dimensional parts that easily get stuck in traps.

In this thesis, we did not try to immediately solve problems as they occur in a practical environment, but rather build a solid foundation for part feeding algorithms. It is, nevertheless, interesting to see how our algorithms perform in experimental setups. We refer the reader to Section 6.4 for a picture of an in-line vibratory feeder with traps that successfully feeds a stream of parts. The feeder was built by Cheung and Smith, see also [12]. In his thesis, Wiegley [110] reports the results of physical experiments with fences across a conveyor belt related to Chapter 3 of this thesis. Also Rusaw *et al.* [93] report on empirical experiments with fences in the presence of friction.

I hope this thesis will motivate researchers to further explore the field of part feeding, and robotic manipulation in general, from an algorithmic perspective. This will hopefully lead to algorithmic solutions that will be applicable to real-world devices.

APPENDIX A

Fence designs for arbitrary parts

In this appendix, we prove completeness of fence design for arbitrary polygonal parts. The proof shows the details of the MOVE, SHIFT and REDUCE framework for the three different classes of parts that are distinguished in Section 3.5.

A.1 Acyclic short environments

In this section, we show how to use the MOVE, SHIFT and REDUCE actions to orient a part for which either the left or the right cycle of the stable equilibria is $|\Sigma|$, and no left or right environment is longer than $\pi/2$ in that cycle. Without loss of generality, we may assume that the right cycle is $|\Sigma|$, and the right environments are shorter than $\pi/2$. In this section, we only use reorientations of the jaw in the angular interval $(0, \pi/2)$, which automatically leads to valid fence designs. Since $\lambda < \pi/2$, the SHIFT and MOVE push plans only use reorientations in the angular interval $(0, \pi/2)$. The techniques of this section are also applicable if the left environments are shorter than $\pi/2$. In that case the reorientations of the jaw are in the angular interval $(-\pi/2, 0)$.

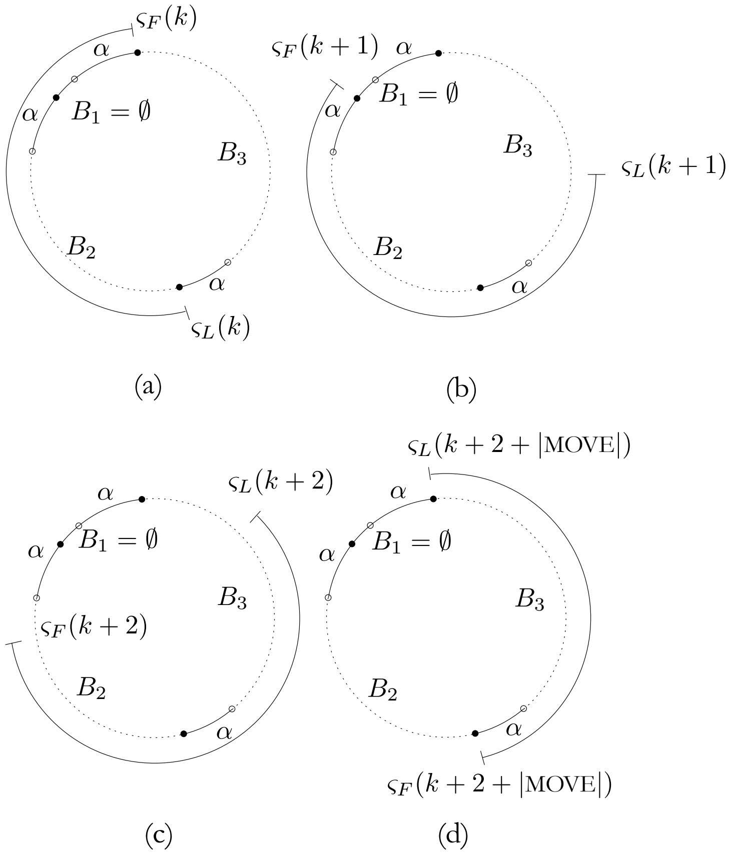
We start with a MOVE. This assures that all possible orientations are in Σ^λ. Let $[\varsigma_F(k = |\text{MOVE}|), \varsigma_L(k = |\text{MOVE}|)]$ be the interval of possible orientations. If $M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)] = 0$, there is only one possible orientation, and we are done. If $M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)] > 0$ we have to reduce the measure of the interval of possible orientations. The goal of the reduce operation is to first develop a plan which maps $\varsigma_F(k)$ onto an orientation in Σ^λ and $\varsigma_L(k)$ onto an orientation not in Σ^λ, with $M_{\Sigma^\lambda}[\varsigma_F(\tilde{k}), \varsigma_L(\tilde{k})] = M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)]$—$\tilde{k}$ denotes the length of the extended plan. We continue this plan with a SHIFT. The SHIFT results in an interval $[\varsigma_F(\tilde{k}+1), \varsigma_L(\tilde{k}+1)]$ of smaller measure, and therefore completes a REDUCE. After another MOVE operation, the resulting interval of possible orientations has again orientations in Σ^λ. We repeat this procedure of reducing the measure until there is just one possible orientation left. Figure A.1 gives an example of application of one step of the general framework. Figure A.1(a) shows the interval $[\varsigma_F(k), \varsigma_L(k)]$ of possible orientations before the reduce. It consists of three orientations in Σ^λ. The measure of $[\varsigma_F(k), \varsigma_L(k)]$ is two. Figure A.1(b) shows the situation after augmenting the plan with $\lambda + \mu$ (SHIFT). The orientations are appropriately reoriented. In Figure A.1(c) the measure of $[\varsigma_F(k+2), \varsigma_L(k+2)]$ is one, due to appending $\lambda + \epsilon$ (SHIFT) to the plan. Finally, a MOVE gives $[\varsigma_F(k+2+|\text{MOVE}|), \varsigma_L(k+2+|\text{MOVE}|)$. The orientations are in Σ^λ again. This situation is depicted in Figure A.1(d).

The main body of the reduce is based on the difference of subsequences $B_{\sigma_i} = \sigma_{i+1}, \ldots, \sigma_{i'-1}$, such that $[\sigma_i, \sigma_{i'}]$ is a subsequence of Σ, $\sigma_i, \sigma_{i'} \in \Sigma^\lambda$, and $B_{\sigma_i} \cap \Sigma^\lambda = \varnothing$. The sequence B_{σ_i} is so to say an ordered set of right environments *between* two orientations with rightenvironments of length λ.

Let B_{σ_i} and B_{σ_j}, subsequences of Σ, be given. We define an order on the sequences B_{σ_i} and B_{σ_j}. Let $\text{init}(B_{\sigma_i})$ and $\text{init}(B_{\sigma_j})$ be the subsequences of B_{σ_i} and B_{σ_j} obtained by repeatedly removing the last elements σ and σ' if $|r(\sigma)| = |r(\sigma')|$. As a result, the right environments of the last orientations in $\text{init}(B_{\sigma_i})$ and $\text{init}(B_{\sigma_j})$ have different lengths. The order on the sequences B_{σ_i} and B_{σ_j} is as follows:

Definition A.1 *Let $B_{\sigma_i} \neq B_{\sigma_j}$. Let λ_i and λ_j be the length of the longest right environment of any stable equilibrium in $\text{init}(B_{\sigma_i})$ and $\text{init}(B_{\sigma_j})$ respectively. Let k_i be the number of orientations $\sigma \in \text{init}(B_{\sigma_i})$ with $r(\sigma) = \lambda_i$ and let k_j be the number orientations $\sigma \in \text{init}(B_{\sigma_j})$ with $r(\sigma) = \lambda_j$. Then $B_{\sigma_i} \prec B_{\sigma_j}$ if*

- *$B_{\sigma_j} = \varnothing$, or*
- *$B_{\sigma_j} \neq \varnothing$ and $\lambda_i > \lambda_j$, or*
- *$B_{\sigma_j} \neq \varnothing$ and $\lambda_i = \lambda_j$ and $k_i > k_j$,*
- *$B_{\sigma_j} \neq \varnothing$ and $\lambda_i = \lambda_j$ and $k_i = k_j$ and $\text{trail}_{\sigma_i} \prec \text{trail}_{\sigma_j}$, where trail_{σ_i} and trail_{σ_j} are the trailing subsequences of $\text{init}(B_{\sigma_i})$ and $\text{init}(B_{\sigma_j})$ starting right after the last orientations in $\text{init}(B_{\sigma_i})$ and $\text{init}(B_{\sigma_j})$ with right environments of length $\lambda_i = \lambda_j$.*



(a) (b)

(c) (d)

Figure A.1 The general reduce.

It follows that $B_{\sigma_i} \neq B_{\sigma_j}$ if and only if $B_{\sigma_i} \prec B_{\sigma_j}$ or $B_{\sigma_j} \prec B_{\sigma_i}$. Furthermore, $\prec$ is transitive and antisymmetric. Suppose now that for an interval of possible orientations after k fences, $[\varsigma_F(k), \varsigma_F(k)]$, $\varsigma_F(k), \varsigma_L(k) \in \Sigma^{\lambda}$, we have $B_{\varsigma_L(k)} \prec B_{\varsigma_F(k)}$. Lemma A.2 gives us that we can find a sequence of k'' fences such that $\varsigma_F(k+k'') \in B_{\varsigma_F(k)}$, and $\varsigma_F(k+k') \notin \text{init}(B_{\varsigma_F(k)})$, and $\varsigma_L(k+k') \in \text{init}(B_{\varsigma_L(k)})$. This lemma is the heart of the plan to REDUCE the measure of $[\varsigma_F(k), \varsigma_F(k)]$.

Lemma A.2 *Let* $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$ *be the interval of orientations. Let* $\varsigma_F(k), \varsigma_L(k) \in \Sigma^{\lambda}$ *and* $B_{\sigma_j} \prec B_{\sigma_i}$. *There exists a fence design such that* $\varsigma_F(\tilde{k}) \in B_{\varsigma_F(k)}$, *and* $\varsigma_F(\tilde{k}) \notin \text{init}(B_{\sigma_i})$, *and* $\varsigma_L(\tilde{k}) \in \text{init}(B_k)$, *with* $\tilde{k}$ *the length of the resulting design.*

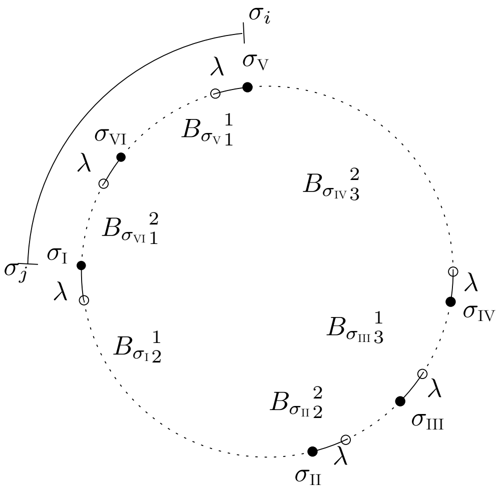
Proof: We are given that $B_{\varsigma_L(k)} \prec B_{\varsigma_F(k)}$. We know that $|r(\varsigma_F(k))| = |r(\varsigma_F(k))|$. Let $\sigma_i = \varsigma_F(k)$, and $\sigma_j = \varsigma_L(k)$. We apply a shift. Now $\varsigma_F(k+1) = \sigma_{i+1}$ and $\varsigma_L = \sigma_{j+1}$ The rest of the plan: reorient the jaw by $(r(\varsigma_F) + \epsilon)$ and push, until we get a design of length $\tilde{k}$, with $\tilde{k}$ the shortest design such that $\varsigma_F(\tilde{k}) \notin \text{init}(B_{\varsigma_F(k)})$. This fairly simple fence design indeed maps $\varsigma_F(k) = \sigma_i$ onto an orientation not in $\text{init}(B_{\sigma_i})$, while $\varsigma_L(k) = \sigma_j$ is mapped onto an orientation in $\text{init}(B_{\varsigma_L(k)})$. We verify this for each item of Definition A.1. If $B_{\sigma_i} = \varnothing$, the proof is trivial. If $B_{\sigma_i} \neq \varnothing$ and $\lambda_{\sigma_j} > \lambda_{\sigma_i}$, then $r(\varsigma_F(k')) < \lambda_{\sigma_j}$ for all $k < k' < \tilde{k}$. If $B_{\sigma_i} \neq \varnothing$ and $\lambda_{\sigma_j} = \lambda_{\sigma_i}$, but $k_j > k_j$, then $\varsigma_L(\tilde{k})$ stays on, or before the last orientation with right environment of length λ_j, in B_{σ_j}. If $B_{\sigma_i} \neq \varnothing$ and $\lambda_j = \lambda_i$ and $k_j = k_i$, then both $\varsigma_F(\tilde{k})$ and $\varsigma_L(\tilde{k})$ are at the k_j'th (and thus also k_i'th) right environments of length $\lambda_j = \lambda_i$ after a number of pushes. Since $B_{\sigma_j} \prec B_{\sigma_i}$, the proof follows inductively from the order of the trailing sequences. □

Lemma A.3 gives us that from the situation of Lemma A.2 we can rather easily reduce the measure of the sequence of possible orientations.

Lemma A.3 *Let* $[\sigma_i, \sigma_{i+m}]$ *and* $[\sigma_j, \sigma_{j+m}]$ *be two disjoint subsequences of orientations in* σ *for which* $r(\sigma_{i+t}) = r(\sigma_{j+t})$ *for* $t \in \{0, \ldots, m\}$. *Let* $\varsigma = \sigma_{i+t} \in [\sigma_i, \sigma_{i+m}]$, *and* $\varsigma' = \sigma_{j+t'} \in [\sigma_j, \sigma_{j+m}]$ *with* $t > t'$. *There is a fence design which maps* ς *onto* σ_{i+m+1}, *and* ς' *onto an orientation in* $[\sigma_j, \sigma_{j+m}]$

Proof: A reorientation of the push direction of $r(\sigma_{i+t}) + \epsilon$ will map ς onto σ_{i+t+1}. We show that ς' is mapped onto an orientation in $[\sigma_{j+t'}, \sigma_{j+t}]$. Suppose that, on the contrary, ς' is mapped onto an orientation beyond σ_{j+t}, this implies that a reorientation of the jaw of $r(\sigma_{i+t}) + \epsilon$ maps ς' beyond the right environment of σ_{j+t}. Clearly this can only be the case if $t' \geq t$ which contradicts the assumption that $t > t'$. Incrementing t, and repeating the argument completes the proof. □

Combining Lemma A.2, Lemma A.3 and a MOVE gives a recipe for a REDUCE plan, given that $B_{\varsigma_L(k)} \prec B_{\varsigma_F(k)}$. In the following, we know show that we can always get to a such a sequence of possible orientations. Repeatedly using (SHIFT, MOVE), we can map an interval of possible orientations $[\sigma_i, \sigma_j], \sigma_i, \sigma_j \in \Sigma^\lambda$ onto another interval $[\sigma_{i'}, \sigma_{j'}], \sigma_{i'}, \sigma_{j'} \in \Sigma^\lambda$. We want to determine the number of repetitions of (SHIFT, MOVE) which leads to $[\sigma_{i'}, \sigma_{j'}], \sigma_{i'}, \sigma_{j'} \in \Sigma^\lambda$ having $B_{\sigma_{j'}} \prec B_{\sigma_{i'}}$. In order to prove that such number exists, we observe the following: The measure M_{Σ^λ} of $[\sigma_i, \sigma_j]$ divides the set of $\{B_\sigma | \sigma \in \Sigma^\lambda\}$ into equivalence classes. An example of these classes is depicted in Figure A.2. The elements of one equivalence class lie $M_{\Sigma^\lambda}([\sigma_i, \sigma_j])$ apart. We shall show that at least one



σ_i
σ_j
λ
σ_{V}
σ_{VI}
σ_{I}
σ_{II}
σ_{III}
σ_{IV}
$B_{\sigma_{\mathrm{V}}1}^{\,1}$
$B_{\sigma_{\mathrm{IV}}3}^{\,2}$
$B_{\sigma_{\mathrm{VI}}1}^{\,2}$
$B_{\sigma_{\mathrm{III}}3}^{\,1}$
$B_{\sigma_{\mathrm{I}}2}^{\,1}$
$B_{\sigma_{\mathrm{II}}2}^{\,2}$

Figure A.2 The measure of $[\sigma_i, \sigma_j]$ is two. The number of right environments with length λ is 6. $[\sigma_i, \sigma_j]$ generates two equivalence classes in the set of B's. These equivalence classes are denoted with the superscript 1 and 2.

equivalence class must contain a pair of orientations which breaks the symmetry of the push function.

We formally define the equivalence classes. Let $M_{\Sigma^\lambda}([\sigma_i, \sigma_j])$ denote the measure of the possible orientations. Let I be the principal ideal generated by $M_{\Sigma^\lambda}([\sigma_i, \sigma_j])$ in $\mathbb{Z}_{|\Sigma^\lambda|}$. The cosets of $\mathbb{Z}_{|\Sigma^\lambda|}/I$ correspond to the equivalence classes of the orientations in Σ^λ, generated by the measure of $M_{\Sigma^\lambda}([\sigma_i, \sigma_j])$. The cosets are sets of orientations in Σ^λ such that the measure of an interval of the same coset is $K \cdot M_{\Sigma^\lambda}([\sigma_i, \sigma_j])$, for an integer K.

Lemma A.4 *If a coset $s + I$ of $\mathbb{Z}_{|\Sigma^\lambda|}/I$ does not have complete identical elements, then there is a configuration of the interval $[\sigma_i, \sigma_j]$ of possible orientations, such that $B_{\sigma_j} \prec B_{\sigma_i}$.*

Proof: Since $\mathbb{Z}_{|\Sigma^\lambda|}$ has a finite number of elements, so does each coset of $\mathbb{Z}_{|\Sigma^\lambda|}/I$. The elements of a coset $s + I$ by assumption, are not identical, but $s + I$ is cyclic since **for all** $s \in \mathbb{Z} : s = s \cdot |\Sigma^\lambda|$ for s in $\mathbb{Z}_{|\Sigma^\lambda|}$. The rest of the proof follows from the transitivity, and the antisymmetry of $\prec$. □

The following lemma states the existence of a coset that contains a pair of non-identical elements.

Lemma A.5 *There is a coset* $s+I$ *of* $\mathbb{Z}_{|\Sigma^\lambda|}/I$ *that does not have identical elements.*

Proof: If all cosets of $\mathbb{Z}_{|\Sigma^\lambda|}/I$ have identical elements, then the right cycle of Σ is $M_{\Sigma^\lambda}([\sigma_i,\sigma_j])$. Since the right cycle of Σ is not determined by Σ^λ, and the environments $B_\sigma(\sigma \in \Sigma^\lambda)$ are divided into identical equivalence classes. There is no difference between any pair of right-environments that are $M_{\Sigma^\lambda}([\sigma_i,\sigma_j])$ apart. However, the right cycle is $|\Sigma|$. This is a contradiction and there has to be a coset of $\mathbb{Z}_{|\Sigma^\lambda|}/I$ that has a pair of non-identical elements. □

To use this asymetric property, we have to be able to shift the interval of possible orientations such that $B_{\sigma_j} \prec B_{\sigma_i}$.

Lemma A.6 *Let* $[\sigma_i,\sigma_j]$ *be an interval of possible orientations. There is an fence design which maps* $[\sigma_i,\sigma_j]$ *onto* $[\sigma_{i'},\sigma_{j'}]$*,* $M_{\Sigma^\lambda}[\sigma_{i'},\sigma_{j'}] = M_{\Sigma^\lambda}[\sigma_i,\sigma_j]$ *such that* $B_{\sigma_{j'}} \prec B_{\sigma_{i'}}$.

Proof: Lemma A.5 gives us that there is a pair $\sigma_{i'},\sigma_{j'}, M_{\Sigma^\lambda}[\sigma_{i'},\sigma_{j'}] = M_{\Sigma^\lambda}[\sigma_i,\sigma_j]$, with $B_{\sigma_{j'}} \prec B_{\sigma_{i'}}$. The push plan (SHIFT, MOVE)^K, with $K = M_{\Sigma^\lambda}[\sigma_i,\sigma_{i'}]$ maps σ_i onto $\sigma_{i'}$ and σ_j onto $\sigma_{j'}$ and is a valid fence design. This completes the proof. □

We now put the building blocks together. The following theorem gives us fence designs to orient parts without large right environments and right cycle $|\Sigma|$.

Theorem A.7 *Let* P *be a part with possible (stable) orientations* Σ*. Let the right cycle of* Σ *be* $|\Sigma|$*. If for all* $\sigma \in \Sigma : |r(\sigma)| < \pi/2$*, then we can orient the* P.

Proof: The following fence design orients the part.

```
1.  MOVE
2.  let $[\varsigma_F(k), \varsigma_L(k)]$ be the interval of possible orientations.
3.  while $M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)] > 0$ do
    3.1   $(\text{SHIFT}, \text{MOVE})^K$, such that
          $B_{\varsigma_L(k+K\cdot|\text{SHIFT,MOVE}|)} \prec B_{\varsigma_F(k+K\cdot|\text{SHIFT,MOVE}|)}$
    3.2   REDUCE
    3.3   $k = k + K \cdot |\text{SHIFT}, \text{MOVE}| + |\text{REDUCE}|$
```

□

This approach also works for parts for which the left environments have the same properties, the reorientations are then in the angular interval $(-\pi/2, 0)$.

A.2 Cyclic environments

In this section we treat parts for which both the left and the right cycle of the possible orientations Σ are strictly smaller than $|\Sigma|$. For these parts, the symmetry of the push function is neither broken by the left environments nor by the right environments, but by the combination of the left and right environments.

A.2.1 Preliminaries

Let c_l denote the left cycle of Σ, and c_r the right cycle of Σ. Let us assume that the period of the push function is 2π, which implies that the least common multiple of c_l and c_r is $|\Sigma|$.
We already noticed that the existence of left or right environments which are longer than $\pi/2$ makes it less straightforward to give push plans which are fence designs as well. Long environments raise the need of large reorientations of the jaw. Such reorientations have to be carefully embedded into a push plan in order to satisfy the fence design constraints.
We start with identifying a number of properties of the push function for parts with cyclic left and right environments, and analyze the possible existence of large environments. Since the right cycle is less than $|\Sigma|$, it follows that for each $\sigma_i \in \Sigma$, there is an σ_j, such that $i \neq j$ and $|r(\sigma_j)| = |r(\sigma_i)|$. More specific, this implies that if there is an σ_i with $|r(\sigma_i)| \geq \pi/2$, there is an σ_j, such that $i \neq j$ and $|r(\sigma_j)| = |r(\sigma_i)|$. Since the period of the push function is 2π, this means that every other orientation σ_t, $t \notin \{i, j\}$, for which $|r(\sigma_t)| \geq \pi/2$, satisfies $|r(\sigma_t)| = |r(\sigma_i)|$. The following lemma gives a result on existence of left or right environments longer than equal $\pi/2$.

Lemma A.8 *Let P be a part. Let $\Sigma = \sigma_1, \ldots, \sigma_{|\Sigma|}$ be the set of $|\Sigma|$ possible stable orientations of P. Let c_l denote the left cycle of Σ, and c_r denote the right cycle of Σ. If $c_l, c_r < |\Sigma|$ and there is an stable equilibrium σ for which $|l(\sigma)| \geq \pi/2$, then the set of stable equilibria $\{\sigma' \in \Sigma \ : \ l(\sigma')| = |l(\sigma)| \geq \pi/2\}$ has cardinality two or three. Furthermore, there is no stable equilibrium σ' for which $|r(\sigma')| \geq \pi/2$.*

Proof: We know that $|l(\sigma_t)| = |l(\sigma_{i+c_l})|$, for all t. Since the left cycle c_l of Σ is less than $|\Sigma|$, each left environment length occurs at least twice. Since the period of the push function is 2π, and each left and right environment has non-zero length, there are at most three (left or right) environments with length greater than or equal $\pi/2$. Suppose now, that there is a right environment with length greater than or equal $\pi/2$. Since the right cycle of Σ is less than $|\Sigma|$ as well, there must be a second right environment with length greater than or equal $\pi/2$, which together with the at least two large left environments does not fit the period of the push function. This contradicts the assumption that there is a right environment with length greater than or equal $\pi/2$, and completes the proof. $\square$

The lemma also holds for a right environment longer than or equal $\pi/2$, thus either the longest left environments, or the longest right environments are longer than $\pi/2$, or none of the left or right environments is longer than $\pi/2$. We assume the following:

Assumption A.9 *Throughout this section we assume, without loss of generality, that there is no orientation* $\sigma \in \Sigma$ *such that* $|r(\sigma)| \geq \pi/2$.

A.2.2 The reduce

The push plan for a part with a left and right cyclic push function is an extension of the techniques to orient parts with acyclic push functions and short environments, which we discussed in Section A.1. Recall that we developed a plan to orient a part with right cycle $|\Sigma|$. In this section, however, the cycle of the right environments unequals $|\Sigma|$. The scheme of Section A.1 might look useless at first glance; if we apply the scheme, we can only reduce the measure of the interval of possible orientations up to the cycle of the right environments. At this point, $|r(\sigma_{i+\bar{\imath}})| = |r(\sigma_{j+\bar{\imath}})|$, for any $\bar{\imath}$, but the part is not oriented up to symmetry yet.

The reduce we will develop, REDUCE$_{dc}$, further orients the part. REDUCE$_{dc}$uses REDUCE for subplans. Since solely the right environments of the parts push function do not capture the part's asymmetry, REDUCE$_{dc}$ exploits the difference of the left environments as well. To understand REDUCE$_{dc}$ we define a second order, $\prec_{dc}$, on intervals of possible orientations.

Definition A.10 *Let* $[\sigma_i, \sigma_j]$ *denote an interval of stable orientations of a part* P. *Let* $|l(\sigma_{i+1})| \neq |l(\sigma_{j+1})|$. *We define* $\sigma_i \prec_{dc} \sigma_j$ *if* $|l(\sigma_{i+1})| < |l(\sigma_{j+1})|$.

Consider the interval of possible orientations $[\varsigma_F(k), \varsigma_L(k)]$. We shall show that we can define a reduce operation which first decreases the number of orientations in the interval of possible orientations and then reduces the right measure of interval of possible orientations of the part.
Our first goal is to shift interval $[\varsigma_F(k), \varsigma_L(k)]$ of possible orientations. We use the plan SHIFT^K, with K such that, at the end, the interval $[\varsigma_F(k+K \cdot |\text{SHIFT}|), \varsigma_L(k+K \cdot |\text{SHIFT}|)]$ of possible orientations satisfies $\varsigma_F(k + K \cdot |\text{SHIFT}|) \prec_{dc} \varsigma_L(k + K \cdot |\text{SHIFT}|)$. Such a K indeed exists. The proof is similar to the equivalence class discussion in Section A.1 and is not repeated here.

Lemma A.11 *Let $[\sigma_i, \sigma_j]$ be an interval of possible orientations. There is a fence design which maps $[\sigma_i, \sigma_j]$ onto $[\sigma_{i'}, \sigma_{j'}]$, with the same number of stable equilibria, but $\sigma_{i'} \prec_{dc} \sigma_{j'}$.*

Next, we will define a reduce for double cyclic push functions, REDUCE_{dc}, which actually decreases the right measure of the interval of possible orientations.
Let $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j], \varsigma_F(k) \prec_{dc} \varsigma_L(k)$ denote the current interval of possible orientations. Firstly, we map $\varsigma_F(k)$ onto $\varsigma_F(k+1) = \sigma_{i+2}$, and $\varsigma_L(k)$ onto $\varsigma_L(k) = \sigma_{j+1}$. The required reorientation of the jaw to accomplish this is $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| + \epsilon$.
The size of the reorientation of the jaw determines whether the reorientation is implemented by a left or a right fence. We continue the discussion for the case $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| + \epsilon < \pi/2$. The other case is treated in the subsequent subsection.

Small reorientations of the jaw

In this subsection we show how to reduce the right measure of the interval $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j], \varsigma_F(k) \prec_{dc} \varsigma_L(k)$ of possible orientations. We are constructing a subplan which starts with a reorientation of the jaw of $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| + \epsilon < \pi/2$ which yields the interval $[\varsigma_F(k+1) = \sigma_{i+2}, \varsigma_L(k) = \sigma_{j+1}]$. We extend REDUCE_{dc} as follows: we continue with $(\text{SHIFT})^{K'}$, such that $\varsigma_F(k + 1 + K' \cdot |\text{SHIFT}|)$ is an orientation in Σ^λ. We finish with another SHIFT, to reduce the right measure of the interval of possible orientations. The next lemma gives that at the end of REDUCE_{dc}, $[\varsigma_F(k + |\text{REDUCE}_{dc}|), \varsigma_L(k + |\text{REDUCE}_{dc}|)]$, which is the image of $[\varsigma_F(k), \varsigma_L(k)]$, is such that the measure $M_{\Sigma^\lambda}([\varsigma_F(k + |\text{REDUCE}_{dc}|), \varsigma_L(k + |\text{REDUCE}_{dc}|)]) < M_{\Sigma^\lambda}([\varsigma_F(k), \varsigma_L(k)])$

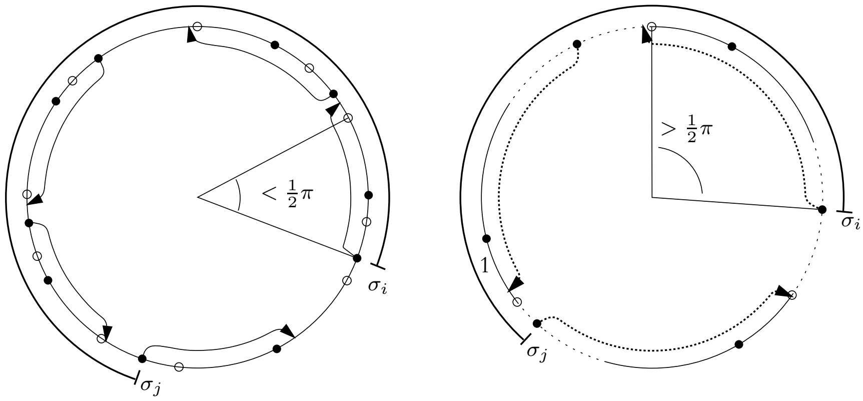
Lemma A.12 *Let* $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$ *denote the interval of stable orientations after k fences. Let* $|l(\sigma_{i+1})| < |l(\sigma_{j+1})|$. *After extending the design with* REDUCE$_{dc}$*, the new interval* $[\varsigma_F(k + |\text{REDUCE}_{dc}|), \varsigma_L(k + |\text{REDUCE}_{dc}|)]$ *of possible orientations has* $M_{\Sigma^\lambda}[\varsigma_F(k + |\text{REDUCE}_{dc}|), \varsigma_L(k + |\text{REDUCE}_{dc}|)] < M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)]$.

Proof: Before the reorientation of $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| + \epsilon$, we have $[\sigma_i, \sigma_j]$, with $i \equiv j \pmod{|\Sigma|/c_r}$, but not $i \equiv j \pmod{|\Sigma|}$. The number of stable equilibria in the interval of possible orientations is $j - i + 1 \pmod{|\Sigma|}$. After application of the jaw, the number of stable equilibria in the interval of possible orientations is less than $j-i+1 \pmod{|\Sigma|}$. If $\sigma_{i+1} \in \Sigma^\lambda$, and the number of stable equilibria in the interval of possible orientations is less than $j - i + 1 \pmod{|\Sigma|}$, then we reduced the right measure. Otherwise, we have to show that the plan $\text{SHIFT}^{K'}$, such that $\sigma_{i+1+K'} \in \Sigma^\lambda$ maintains the property that the number of orientations in the interval of possible orientations is less than $j-i+1 \pmod{|\Sigma|}$. Suppose now that at a certain point during the execution of $(\text{SHIFT})^K$, the number of orientations in the interval of possible orientations becomes larger than or equal to $j - i + 1$. Before the last push, the number of orientations in the interval of possible orientations was less than $j - i + 1$. Let $[\sigma_{i'}, \sigma_{j'}]$ denote this interval of possible orientations, thus $j' - j < (i' - i)$. SHIFT $(= |r(\sigma_{i'})| + \epsilon)$ maps $[\sigma_{i'}, \sigma_{j'}]$ onto $[\sigma_{i'+1}, \sigma_{j''}]$, with $j'' - j' \geq (i' - i) + 1$, and $j' - j < (i' - i)$, thus $j'' > j' + 1$. We know that for any t, $|r(\sigma_{i+t})| = |r(\sigma_{j+t})|$, which implies that there is an orientation $j''' \in (j', j'']$ such that $r(j''') = r(i')$. This means that the last SHIFT, which mapped $\sigma_{j'}$ onto $\sigma_{j''}$, involves a reorientations of the jaw of at least $|r(\sigma_{i'})| + |r(\sigma_{j'})| > |r(\sigma_{i'})| + \epsilon$. ☐

Large reorientations of the jaw

If we drop the assumption that there is a configuration of the interval of possible orientations $[\sigma_i, \sigma_j]$, such that $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| < \pi/2$, the REDUCE$_{dc}$ plan from the previous paragraph does not correspond to a valid fence design. We will slightly alter REDUCE$_{dc}$ to overcome this problem. The following theorem gives us that in this special case the length of the longest left environment, $\reflectbox{$\lambda$} < \pi/2$. This property will turn out to be useful. Recall that we assumed that $\lambda < \pi/2$ as well (Assumption A.9).

Lemma A.13 *Let* P *be a part. Let* $\Sigma = \{\sigma_1 \ldots \sigma_{|\Sigma|}\}$ *be the set of* $|\Sigma|$ *possible stable orientations of* P. *Let* c_l *denote the left cycle of* Σ, *and* c_r *denote the right cycle*



(a) (b)

Figure A.3 (a) The situation where we can skip one special large right environment (1). (b) The situation where we can use a change of fence side, and continue with the mirrored strategy.

of Σ. *Let* $c_l, c_r < |\Sigma|$. *Let* $ʎ = \max_{i\in\{1\ldots|\Sigma|\}} |l(\sigma_i)|$. *If there is an* $\bar\imath$ *such that* $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| > \pi/2$ *for all i, for which* $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| < |r(\sigma_{i+(\bar\imath|\Sigma|/c_r)})| + |l(\sigma_{i+1+(\bar\imath|\Sigma|/c_r)})| + |r(\sigma_{i+1+(t|\Sigma|/c_r)})|$, *then* $ʎ < \pi/2$.

Proof: Let us assume, on the contrary that $ʎ > \pi/2$. According to Lemma A.8, there are at least two left environments which have length $ʎ$, say $l(\sigma_{\mathrm{I}})$ and $l(\sigma_{\mathrm{II}})$ with $\sigma_{\mathrm{I}}, \sigma_{\mathrm{II}} \in \Sigma$. We will show that for each $t > 0$, there is an i, such that $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| < |r(\sigma_{i+(t|\Sigma|/c_r)})| + |l(\sigma_{i+1+(t|\Sigma|/c_r)})| + |r(\sigma_{i+1+(t|\Sigma|/c_r)})|$, and $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| < \pi/2$. Let $t > 0$ be given. We choose j and $\tilde{\jmath}$ such that $\sigma_{j+1+(t|\Sigma|/c_r)}$ coincides with σ_{I}, and $\sigma_{\tilde{\jmath}+1+(t|\Sigma|/c_r)}$ coincides with σ_{II}. Since $c_r \neq c_l$, $|\Sigma|/c_r \neq |\Sigma|/c_l$, and clearly also $|l(\sigma_{j+1})|, |l(\sigma_{\tilde{\jmath}+1})| < |l(\sigma_{j+1+(t|\Sigma|/c_r)})|$. If $|r(\sigma_j)| + |l(\sigma_{j+1})| + |r(\sigma_{j+1})| < \pi/2$, we let $i = j$. Otherwise, we shall show that $|r(\sigma_{\tilde{\jmath}})| + |l(\sigma_{\tilde{\jmath}+1})| + |r(\sigma_{\tilde{\jmath}+1})| < \pi/2$. In order to have $c_l, c_r < |\Sigma|$, there have to be more than four stable equilibria. Therefore, if $\{\sigma_{\tilde{\jmath}}, \sigma_{\tilde{\jmath}+1}\} \cap \{\sigma_j, \sigma_{j+1}\} = \varnothing$, then $|r(\sigma_{\tilde{\jmath}})| + |l(\sigma_{\tilde{\jmath}+1})| + |r(\sigma_{\tilde{\jmath}+1})| \leq 2\pi - 2ʎ - |r(\sigma_j)| - |l(\sigma_{j+1})| - |r(\sigma_{j+1})| - |l(\sigma)| - |r(\sigma)| < \pi/2$, with σ a fifth stable equilibrium. It remains to prove that $\{\sigma_{j'}, \sigma_{j'+1}\} \cap \{\sigma_j, \sigma_{j+1}\} = \varnothing$. Firstly, $\sigma_j \neq \sigma_{\tilde{\jmath}}$, and $\sigma_{j+1} \neq \sigma_{\tilde{\jmath}+1}$, because otherwise $\sigma_{\mathrm{I}} = \sigma_{\mathrm{II}}$. Secondly, $\sigma_j \neq \sigma_{\tilde{\jmath}+1}$ and $\sigma_{j+1} \neq \sigma_{\tilde{\jmath}}$, because otherwise σ_j neigbors $\sigma_{\tilde{\jmath}}$, and therefore σ_{I} neighbors σ_{II} which implies that $c_l = 1$ and $c_r = |\Sigma|$. This contradicts the assumption. So, we can let $i = \tilde{\jmath}$ in this case, which completes the proof. $\square$

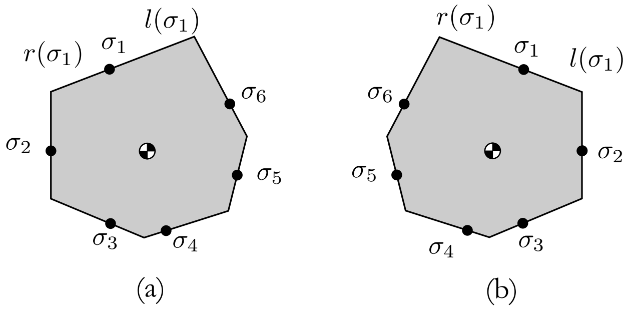


Figure A.4 (a) A part, and (b) its mirror image.

We know that $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| \geq \pi/2$, so $\reflectbox{$\lambda$} < \pi/2$. Furthermore $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| < \pi$, since otherwise the period of the push function is greater than 2π. If we now reorient the jaw by $|r(\sigma_i)| + |l(\sigma_{i+1})| + |r(\sigma_{i+1})| + \epsilon$, and apply the jaw, we reduced the length (but not necessarily the right measure) of the interval of possible orientations. At this point, we have to use reorientations in the angular intervals $(-\pi/2, 0)$ or $(-\pi, -\pi/2)$. The part of the reduce that follows is a *mirrored* version of the reduce presented up till now. We can therefore also continue to orient the mirrored part, and keep in mind that we have to replace the angles by which we reorient the jaw by their negated values. Also, a mirror swaps the values of c_l and c_r, λ and $\reflectbox{$\lambda$}$, and the sets Σ^{λ} and $\Sigma^{\reflectbox{$\lambda$}}$. Figure A.4 depicts a part and its mirror image. We now continue to reduce the inverted right measure of the (mirrored) interval of possible orientations.

If $i \not\equiv j \pmod{|\Sigma|/c_r}$ we continue REDUCE$_{dc}$ by mirror(SHIFT^K), until $\varsigma_L(k + 1 + K \cdot |\text{SHIFT}|) \in \Sigma^{\lambda}$, a final SHIFT reduces the measure, and is valid since (the new) $\lambda < \pi/2$.

We now know $i \equiv j \pmod{|\Sigma|/c_r}$ and try to (further) reduce the mirrored right measure of the interval of possible orientations, using mirrored SHIFT's and REDUCE$_{dc}$'s. Note that if it happens to be that case that the REDUCE$_{dc}$ is again forced to be mirrored, then, after two subsequent mirrors of the reduce, the interval of possible orientations $[\sigma_{i'}, \sigma_{j'}]$ is shorter than the interval of possible orientations $[\sigma_i, \sigma_j]$ before the former mirror. Since two mirrors of the part cancel out, we continue REDUCE$_{dc}$ as if we did not use the two mirrors until the part is oriented.

A.2.3 The final reduce

In this section we derived the following:

Theorem A.14 *Let P be a part. If the P's push functions stable equilibria Σ have a left cycle $c_l < |\Sigma|$ and a right cycle $c_r < |\Sigma|$, then there is a fence design that orients the part up to symmetry.*

Proof: The discussion in this section leads to the following REDUCE$_{dc}$.

```
1. MOVE
2. let $[\varsigma_F(k), \varsigma_L(k)]$ be the interval of possible orientations.
3. while $\varsigma_F(k) \neq \varsigma_L(k)$ do
   3.1   if  there is a pair of orientations $\sigma_i, \sigma_j \in \Sigma^\lambda$,
             with $M_{\Sigma^\lambda}[\sigma_i, \sigma_j] = M_{\Sigma^\lambda}[\varsigma_F(k), \varsigma_L(k)]$
             and $B_{\sigma_j} \prec B_{\sigma_i}$
         then
             3.1.1. $(\text{SHIFT}, \text{MOVE})^K$, $K$ such that $\varsigma_F(k + K \cdot |\text{SHIFT}|) = \sigma_i$,
                    and $\varsigma_F(k + K \cdot |\text{SHIFT}|) = \sigma_j]$
             3.1.2. REDUCE
             3.1.3  $k = k + K \cdot |\text{SHIFT}| + |\text{REDUCE}|$
   3.2.  else
             3.2.1  $\text{SHIFT}^K$, $K$ such that $\varsigma_F(k + K \cdot |\text{SHIFT}|) = \sigma_i$,
                    and $\varsigma_F(k + K \cdot |\text{SHIFT}|) = \sigma_j]$
                    and REDUCE$_{dc}$ is applicable
             3.2.2  REDUCE$_{dc}$; take mirror into account.
             3.2.3  $k = k + K \cdot |\text{SHIFT}| + |\text{REDUCE}_{dc}|$
```

□

A.3 Acyclic environments

We now treat the special cases where some environment (left or right) is longer than $\pi/2$. Without loss of generality we may assume that some right environment

has size $\geq \pi/2$. Unfortunately, the MOVE, SHIFT and REDUCE, as defined in the introduction of this section, no longer only use reorientations of the jaw in the angular interval $(0, \pi/2)$. In this section, we use the same general idea of MOVE, SHIFT and REDUCE, but we have to modify them into feasible fence designs.
We use the following push plans as building blocks:

- Suppose that $[\varsigma_F(k) = \sigma_i, \varsigma_L(k) = \sigma_j]$ $(\sigma_i, \sigma_j \in \Sigma^\lambda)$ is the current interval of possible orientations. We want a SHIFT_l operation which maps $[\varsigma_F(k), \varsigma(k)]$ onto $[\varsigma_F(k + |\text{SHIFT}_l|) \neq \sigma_i, \varsigma_L(k) \neq \sigma_j]$ with equal measure. This is a useful operation when $|\Sigma^\lambda| > 2$. The SHIFT_l will be presented in Section A.3.1.
- We need an operation REDUCE_l which reduces the measure of $[\varsigma_F(k), \varsigma_L(k)]$.

Note that the maximum number of right environments longer than $\pi/2$ is three (otherwise, the period of the push function is larger than 2π). If there is one right environment that is larger than $\pi/2$, we simply use a fence design $(\pi/2 - \epsilon)^{|\Sigma|}$. It remains to prove that for parts with more than one orientation with a right environment longer than $\pi/2$, we can extend this design to a fence design which also orient these parts. Throughout this section, we denote the stable equilibria in Σ^λ by σ_{I}, σ_{II}, and if there is a third large right environment, σ_{III}. We call the possible orientations after the first move, hence after $k > |\text{MOVE}| = |\Sigma|$ fences $\varsigma_{\text{I}}(k)$, $\varsigma_{\text{II}}(k)$, and if applicable, $\varsigma_{\text{III}}(k)$. We assume that $\varsigma_{\text{I}}(|\text{MOVE}|) = \sigma_{\text{I}}$, $\varsigma_{\text{II}}(|\text{MOVE}|) = \sigma_{\text{II}}$, and $\varsigma_{\text{III}}(|\text{MOVE}|) = \sigma_{\text{III}}$. We denote $B_{\sigma_{\text{I}}}$ by B_{I}, $B_{\sigma_{\text{II}}}$ by B_{II} and $B_{\sigma_{\text{III}}}$ by B_{III}.
We will clarify the presented push plans by giving pictures which symbolically show the reorientations of the jaw. The part is represented by the circular representation of its push function. The stable equilibria correspond to discs, the unstable equilibria correspond to circles. Recall that reorientations of the jaw in the angular intervals $(0, \pi/2)$ and $(-\pi, -\pi/2)$ are implemented by a left fence. Reorientations of the jaw in the angular intervals $(-\pi/2, 0)$ and $(\pi/2, \pi)$ are implemented by a right fence. In the pictures, reorientations of the jaw implemented by a left fence correspond to solid arrows. Reorientations of the jaw implemented by of a right fence correspond to dashed arrows. A push plan $\alpha_1, \alpha_2, \alpha_3$ adds three arrows per possible orientation to the figure. The first reorientation is marked by **1**, the second by **2**, and the third by **3**.

We start by moving the set of possible orientations to the long right environments. In Figure A.5 the MOVE is depicted for a part with long environments. In the next subsections we treat the several cases which can occur if there are long right environments.

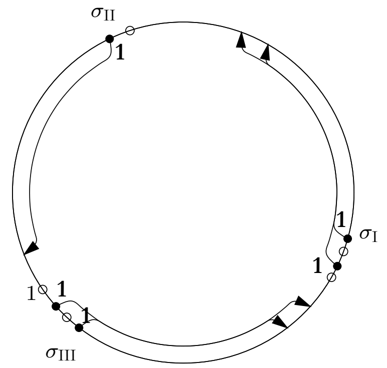


Figure A.5 The MOVE for a part with three long right environments $r(\sigma_{\mathrm{I}}), r(\sigma_{\mathrm{II}})$, and $r(\sigma_{\mathrm{III}})$. In this example, one reorientation of the jaw suffices to MOVE the possible orientations onto $\{\sigma_{\mathrm{I}}, \sigma_{\mathrm{II}}, \sigma_{\mathrm{III}}\}$.

A.3.1 Three long environments

In this section we treat parts of which three orientations have a right environment longer than $\pi/2$. In the three large-right-environment case, we now apply a three phase approach

1. REDUCE$_l$ the three possible orientations to two orientations with a large right environment. This phase thus reduces two large orientations to one. We design the reduce plan such that the last reorientation we use is in the angular interval $(0, \pi/2)$, or $(-\pi, -\pi/2)$, i.e. the last fence was a left fence.
2. SHIFT$_l$ the possible orientations. Depending on the REDUCE$_l$, either one or two shifts are necessary.
3. Apply the REDUCE$_l$ again.

We first present SHIFT$_l$, which is the same in all cases in this section. Let us assume that the length of the design, constructed so far, is k, and the k-th reorientation of the jaw was in the angular interval $(0, \pi/2)$, and the possible orientations are in Σ^λ (the right environments are longer than $\pi/2$). This means that any reorientation in $(0, \pi/2)$ is useless, it does not change the possible orientations. So, in order to give a valid fence design, we have to use at least one reorientation in the angular interval $(\pi/2, \pi)$. Let us, for easy of presentation, assume that $\varsigma_{\mathrm{I}}(k) = \sigma_{\mathrm{I}}$, $\varsigma_{\mathrm{II}}(k) = \sigma_{\mathrm{II}}$, and $\varsigma_{\mathrm{III}}(k) = \sigma_{\mathrm{III}}$. The first reorientation we use is $\max_{\sigma \in \Sigma^\lambda}(|r(\sigma)|) + \epsilon$. After application of the jaw, clearly, $\varsigma_{\mathrm{I}}(k+1) \neq \sigma_{\mathrm{I}}$, $\varsigma_{\mathrm{II}}(k+1) \neq \sigma_{\mathrm{II}}$ and $\varsigma_{\mathrm{III}}(k+1) \neq \sigma_{\mathrm{III}}$. The second reorientation we use is

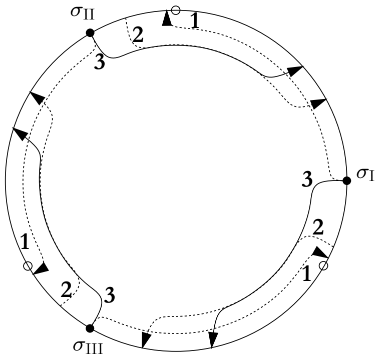


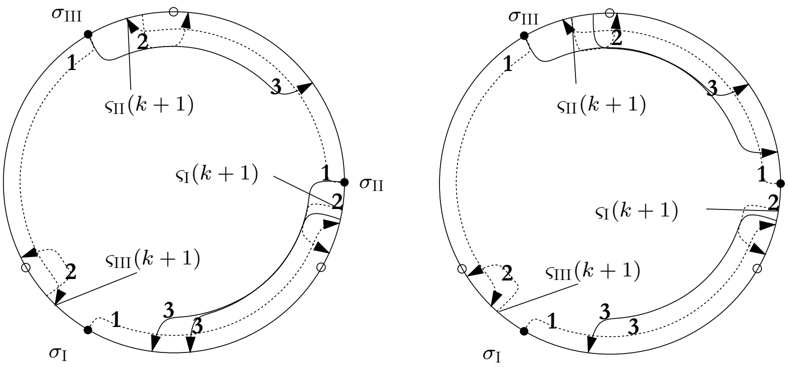
Figure A.6 The SHIFT$_l$ plan for three long right environments is $(\pi/2 + \epsilon), (-(\pi/2 - \epsilon)), (-(\pi/2 + \epsilon))$.

$(-(\pi/2 - \epsilon))$. Since there are three right environments longer than $\pi/2$, and at least three left environments, the possible orientations of the part are now again $\varsigma_{\mathrm{I}}(k + 2) = \sigma_{\mathrm{I}}$, $\varsigma_{\mathrm{II}}(k + 2) = \sigma_{\mathrm{II}}$ and $\varsigma_{\mathrm{III}}(k + 2) = \sigma_{\mathrm{III}}$. However, we can now use reorientations in the angular intervals $(-\pi, -\pi/2)$ and $(-\pi/2, 0)$. The first two reorientations of the shift are implemented by right fences. The third and last reorientation of the jaw in SHIFT$_l$ is $-(\pi/2 + \epsilon)$, we skip the angular intervals between $r(\sigma_{\mathrm{I}})$, $r(\sigma_{\mathrm{II}})$ and $r(\sigma_{\mathrm{III}})$, resulting in a configuration of the possible orientations in which $\varsigma_{\mathrm{I}}(k + 3) = \sigma_{\mathrm{III}}$, $\varsigma_{\mathrm{II}}(k + 3) = \sigma_{\mathrm{I}}$ and $\varsigma_{\mathrm{III}}(k + 3) = \sigma_{\mathrm{II}}$. This reorientation is implemented by a left fence. We have now shifted the possible orientations and can again use reorientations of the jaw in the angular intervals $(0, \pi/2)$, and $(\pi/2, \pi)$. The shift is depicted in Figure A.6.

In the next subsection we present reduce plans which are applicable for parts with three long right environments. Since there are three long right environments, there cannot be a left environment longer than $\pi/2$. If the left cycle of the part is less than $|\Sigma|$, then we can use the techniques of Section A.1. Since we already showed how to orient a part of which both the left and right cycle are less than $|\Sigma|$, we may assume that the right cycle is $|\Sigma|$.

Three long environments of equal length

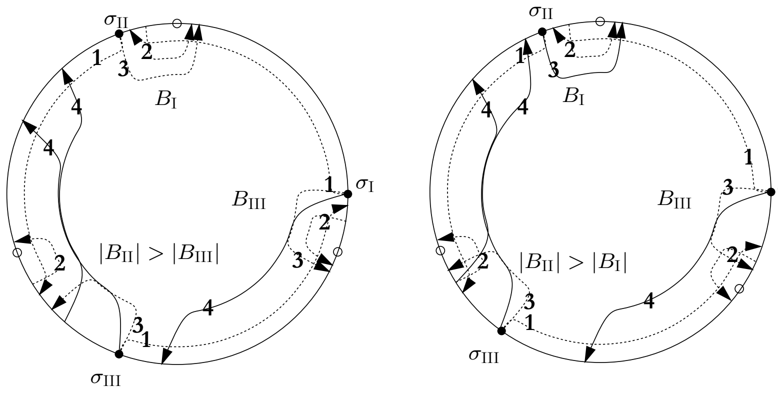
In this section we assume that $|r(\sigma_{\mathrm{I}})| = |r(\sigma_{\mathrm{II}})| = |r(\sigma_{\mathrm{III}})|$. Recall that the right cycle is $|\Sigma|$. Let k be the number of fences used so far, $\varsigma_{\mathrm{I}}(k) = \sigma_{\mathrm{I}}$, $\varsigma_{\mathrm{II}}(k) = \sigma_{\mathrm{II}}$, and $\varsigma_{\mathrm{III}}(k) = \sigma_{\mathrm{III}}$. We have two cases.



(a) (b)

Figure A.7 The REDUCE$_l$ for parts with three long right environments of equal length and $|B_{\rm I}| = |B_{\rm II}| = |B_{\rm II}|$ first break the symmetry by a reorientation in $(\pi/2, \pi)$ the rest of the plan is $-(\varsigma_{\rm II}(k+1) - (\sigma_{\rm II} + |r(\sigma_{\rm II})| + \epsilon))$

1. In the first case, the angular intervals $B_{\rm I}, B_{\rm II}$, and $B_{\rm III}$, between $r(\sigma_{\rm I})$, $r(\sigma_{\rm II})$ and $r(\sigma_{\rm III})$, have exactly the same size. Clearly for $i \in \{{\rm I}, {\rm II}, {\rm III}\}$: $|B_i| + |r(\sigma_i)| < \pi$, and therefore $B_i \cap (\sigma_i + \pi/2, \sigma_i + \pi) = B_i$. In words, with a valid reorientation of the jaw, we can map any possible orientation onto any angle in the angular intervals between the long right environments. Lemma 2.5 gives us that there is a reorientation of the jaw for which one orientation, say $\varsigma_{\rm I}(k)$ is mapped onto a left environment, and another orientation $\varsigma_{\rm II}(k)$ is mapped onto a right environment. The third orientation $\varsigma_{\rm III}(k)$ is mapped either on a left or a right environment. Thus, after the application of the jaw, which is a right fence, we know that *not* $(\varsigma_{\rm I}(k+1) - (\sigma_{\rm I} + |r(\sigma_{\rm I})|)) = (\varsigma_{\rm II}(k+1) - (\sigma_{\rm II} + |r(\sigma_{\rm II})|)) = (\varsigma_{\rm III}(k+1) - (\sigma_{\rm III} + |r(\sigma_{\rm III})|))$. Furthermore, we know that for $i \in \{{\rm I}, {\rm II}, {\rm III}\}(\sigma_i - (\sigma_i + |r(\sigma_i)|)) < \pi/2$. We can now reorient the jaw by $-(\varsigma_{\rm II}(k+1) - (\sigma_{\rm II} + |r(\sigma_{\rm II})| + \epsilon))$ and apply the jaw, which yields $\varsigma_{\rm II}(k+2) = \sigma_{\rm II}$, but $\varsigma_{\rm I}(k+2) \in B_{\rm I}$. The third possible orientation $\varsigma_{\rm III}(k+2)$ is an orientation in $B_{\rm III}$ or $\varsigma_{\rm III}$. The reorientation is in the angular interval $(-\pi/2, 0)$; this is accomplished by a right fence as well. We can now reorient the jaw by $-(\pi/2 + \epsilon)$. An application of the jaw shifts the possible orientation(s) in Σ^λ to the preceding orientation in Σ^λ. The possible orientations which are not in Σ^λ are also shifted to the preceding orientation in Σ^λ. Hence, $\varsigma_{\rm I}(k+3) = \sigma_{\rm I}$, $\varsigma_{\rm II}(k+3) = \sigma_{\rm II}$, and $\varsigma_{\rm III}(k+3) \in \{\sigma_{\rm II}, \sigma_{\rm III}\}$. The third application of the jaw collapses two orientations. This last application is realized by a left fence. We have a feasible reduce. In Figure A.7, the cases for $\varsigma_{\rm III}$ mapped onto $B_{\rm III}$ and $\varsigma_{\rm III}$ mapped onto $\sigma_{\rm III}$ are displayed).



(a) (b)

Figure A.8 The REDUCE$_l$ for parts with three long right environments of equal length and not $|B_{\mathrm{I}}| = |B_{\mathrm{II}}| = |B_{\mathrm{III}}|$ start with $(|r(\sigma_{\mathrm{I}})| + \epsilon), (-(\pi/2 - \epsilon)$, then break the symmetry by a reorientation in $(-\pi/2, 0)$. The rest of the plan is $-(\pi/2 + \epsilon)$.

2. The second case covers the situation in which not $|B_{\mathrm{I}}| = |B_{\mathrm{II}}| = |B_{\mathrm{III}}|$. We break the symmetry using the difference in length of the B_i's. We start with reorientation $|r(\sigma_{\mathrm{I}})| + \epsilon$, this maps a possible orientation $\varsigma(k) = \sigma_i$ onto the next orientation $\varsigma(k+1) = \sigma_{i+1}$. We can now reorient the jaw by $-(\pi/2 - \epsilon)$, after applying the jaw, $\varsigma_{\mathrm{I}}(k+2) = \sigma_{\mathrm{I}}$, $\varsigma_{\mathrm{II}}(k+2) = \sigma_{\mathrm{II}}$, and $\varsigma_{\mathrm{III}}(k+2) = \sigma_{\mathrm{III}}$. With the third reorientation we break the symmetry. We can now reorient in the angular interval $(-\pi/2, 0)$, and we choose the reorientation such that at least one orientation is mapped onto an orientation in B_i $(i \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\})$, and at least one orientation is mapped onto $r(\sigma_i)$ $(i \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\})$. We apply the jaw. We can now with one left fence implement a reorientation of the jaw of $-(\pi/2 + \epsilon)$. This fence maps two possible orientations to one. There are again two cases. See Figure A.8(a) and (b).

Three long environments not of equal length

If not $|r(\sigma_{\mathrm{I}})| = |r(\sigma_{\mathrm{II}})| = |r(\sigma_{\mathrm{III}})|$, then we can reduce number of possible orientations to one using the reduce plan starting with a right fence which maps at least one possible orientation onto itself, and at least one possible orientation onto another orientation. This is accomplished by a reorientation of the jaw by $(\min_{i \in \{\mathrm{I},\mathrm{II},\mathrm{III}\}} r(\sigma_i) + \epsilon)$. After pushing, we reorient the jaw by $-(\pi/2 - \mu)$.

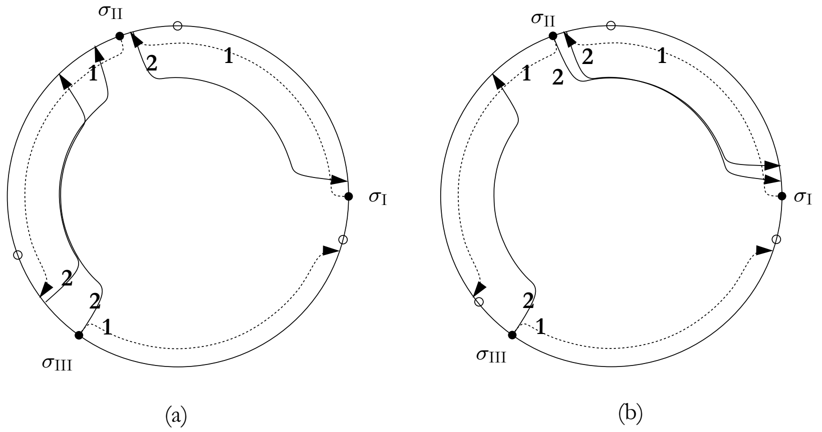


Figure A.9 The REDUCE$_l$ for parts with three long right environments, not of equal length, consists of $(\min_{i \in \{\mathrm{I,II,III}\}} r(\sigma_i) + \epsilon), -(\pi/2 - \mu)$.

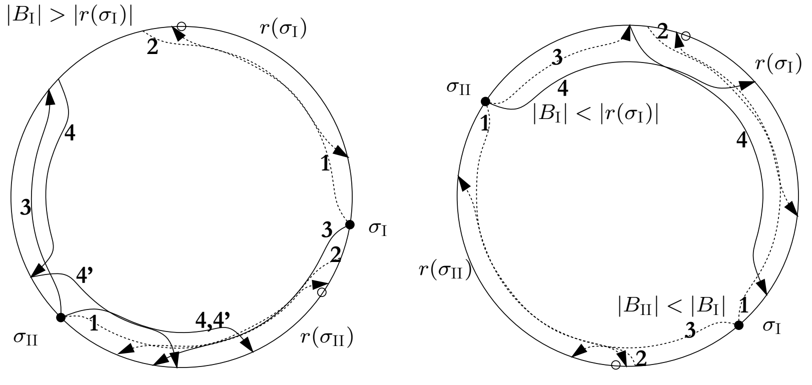
This is accomplished by a left fence which reduces the number of possible orientations to two. In Figure A.9(a) and (b), the REDUCE$_l$ for a unique largest right environment is displayed.

A.3.2 Two long environments

If there are two orientations with long right environments, σ_{I}, σ_{II}. There is no longer a reason to SHIFT$_l$. After a MOVE, there are two possible orientations, and after a reduce, there is only one possible orientation left. In the case of three long environments, the intervals between the long environments, B_{I}, B_{II} and B_{III} are smaller than $\pi/2$. If there are only two long right environments, this is not necessarily true. As a consequence, the reduce for two long right environments is more complicated. We devote the following subsections to present the reduce for the different cases.

Two long environments of equal length

We first give the reduce for the case in which both large right environments have the same size, i.e. $|r(\sigma_{\mathrm{I}})| = |r(\sigma_{\mathrm{II}})|$. The size of the domain of the push-function minus $(r(\sigma_{\mathrm{I}}) \cup r(\sigma_{\mathrm{II}}))$ is smaller than π. If the left-environments have cycle $|\Sigma|$, we can orient the part using the techniques of Section A.1, using reorientations in



(a) (b)

Figure A.10 (a) The REDUCE$_l$ for two long right environments of equal length and $|B_{\text{I}}| > \pi/2$ is $(r(\sigma_{\text{I}}) + \epsilon), -(\pi/2 - \epsilon), -(\pi/2 + \epsilon)$ followed by a MOVE (= **4**). (b) If $|B_{\text{I}}| < |r(\sigma_{\text{I}})|$, REDUCE$_l$ is as follows: $(r(\sigma_{\text{I}}) + \epsilon), -(\pi/2 - \epsilon), -(|B_{\text{II}}| + \epsilon), (-|r(\sigma_{\text{I}})|)$.

the angular interval $(-\pi/2, 0)$. In that case there is at most one left environment with size greater than $\pi/2$. Now suppose that the left cycle is less than $|\Sigma|$, say c_l. Since we already treated the case for which the right cycle is less than $|\Sigma|$ as well, we may now assume that the right cycle is $|\Sigma|$. We now distinguish two cases.

1. We first assume that he lengths of the angular intervals between $r(\sigma_{\text{I}})$ and $r(\sigma_{\text{II}})$ differ. With one reorientation of the jaw of $(|r(\sigma_{\text{I}})| + \epsilon)$ followed by a push, implemented by a right fence, we shift both $\varsigma_{\text{I}}(k)$ and $\varsigma_{\text{II}}(k)$ one orientation to the next stable equilibrium of the part. With another right fence, we shift $\varsigma_{\text{I}}(k+1)$ back to $\varsigma_{\text{I}}(k+2) = \sigma_{\text{I}}$ and $\varsigma_{\text{II}}(k+1)$ back to $\varsigma_{\text{II}}(k+2) = \sigma_{\text{II}}$, using a reorientation of the jaw of $\pi/2 - \epsilon$. Next, the largest angular interval between $r(\sigma_{\text{I}})$ and $r(\sigma_{\text{II}})$ is either larger than $|r(\sigma_{\text{I}})| = |r(\sigma_{\text{II}})|$, or this interval is as long as or shorter than $|r(\sigma_{\text{I}})| = |r(\sigma_{\text{II}})|$.
   Let us first assume that the largest angular interval between $r(\sigma_{\text{I}})$ and $r(\sigma_{\text{II}})$ is longer than $|r(\sigma_{\text{I}})| = |r(\sigma_{\text{II}})|$, and thus larger than $\pi/2$ as well. Let us assume that this angular interval is B_{I}, and thus $\{\sigma_{\text{I}}, B_{\text{I}}, \sigma_{\text{II}}, \}$ is counterclockwisely ordered. The $(k+2)$-th reorientation of the jaw was in the interval $(-\pi/2, 0)$ which gives us that the $(k+3)$-th reorientation of the jaw of $-(\pi/2 + \epsilon)$ is feasible by a push of a left fence. This push results in $\varsigma_{\text{I}}(k+3) = \sigma_{\text{II}}$, and $\varsigma_{\text{II}}(k+3) \in B_{\text{I}}$. In counterclockwise order, $\varsigma_{\text{II}}(k+3)$ is positioned before σ_{II}, so with a MOVE this orientation maps to σ_{II} as well, while the possible orientation ς_{I} remains fixed, and hereby complete a valid and feasible fence design. A picture of this reduce is given in Figure A.10(a).

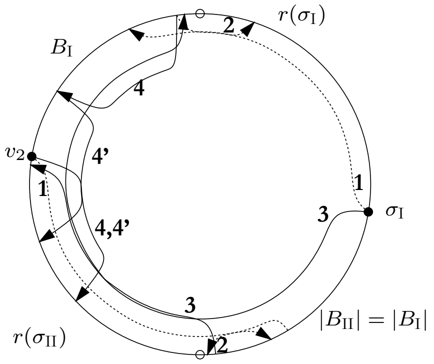


Figure A.11 The REDUCE$_l$ for a part with two long right environments of equal length, and $|B_{\mathrm{I}}| = |B_{\mathrm{II}}|$. The plan starts with a symmetry breaking push after a reorientation fo the jaw in $(\pi/2, \pi)$. The plan continues with $-(\varsigma_{\mathrm{I}}(k+1) - \sigma_{\mathrm{I}})$, $-(\pi - \mu)$ and ends with a MOVE.

If, on the contrary, the largest angular interval between $r(\sigma_{\mathrm{I}})$ and $r(\sigma_{\mathrm{II}})$, B_{I}, is as long as or shorter than $|r(\sigma_{\mathrm{I}})| = |r(\sigma_{\mathrm{II}})|$, we derive a slightly different plan. Since the size of the domain of the push-function minus $(r(\sigma_{\mathrm{I}}) \cup r(\sigma_{\mathrm{II}}))$ is smaller than π, we conclude that $|B_{\mathrm{II}}| < \pi/2$. The sequence $\{\sigma_{\mathrm{I}}, B_{\mathrm{I}}, \sigma_{\mathrm{II}}, B_{\mathrm{II}}\}$ is counterclockwisely ordered. With a reorientation of the jaw of $-(|B_{\mathrm{II}}| + \epsilon)$ which is a push by a right fence, we yield $\varsigma_{\mathrm{I}}(k+3) = \sigma_{\mathrm{II}}$, while $\varsigma_{\mathrm{II}}(k+3) \in B_{\mathrm{I}}$. With a reorientation of the jaw by $(-|r(\varsigma_{\mathrm{I}})|)$ and a push by a left fence, we finally get $\varsigma_F(k+4) = \varsigma_L(k+4) = \sigma_{\mathrm{I}}$. A picture of this case of REDUCE$_l$ is given in Figure A.10(b).

2. In this case, the intervals between the large right environments, B_{I} and B_{II}, have exactly the same size. Since the push function has period 2π, Lemma 2.5 gives us that there has to be a pair of orientations (at least one between the orientations with the large right environments), such that the distance between $\varsigma_{\mathrm{I}}(k+1)$ and σ_{I} is not equal to the distance between $\varsigma_{\mathrm{II}}(k+1)$ and σ_{II}. It is not hard to see that this orientation must be reachable with one push in angular interval $(\pi/2, \pi)$ by a right fence. We extend the design by this reorientation and assume w.l.o.g. that $(\varsigma_{\mathrm{I}}(k+1) - \sigma_{\mathrm{I}}) < (\varsigma_{\mathrm{II}}(k+1) - \sigma_{\mathrm{II}})$. With another right fence, we reorient the jaw by $-(\varsigma_{\mathrm{I}}(k+1) - \sigma_{\mathrm{I}})$ and apply the jaw, which results in $\varsigma_{\mathrm{I}}(k+2) = \sigma_{\mathrm{I}}$ and $\varsigma_{\mathrm{II}}(k+2) \in B_{\mathrm{II}}$. We now reorient the jaw by $-(\pi - \mu)$ and push with a left fence. We yield $\varsigma_{\mathrm{I}}(k+3) = \sigma_{\mathrm{II}}$, and $\varsigma_{\mathrm{II}}(k+3) \in B_{\mathrm{I}}$. With a MOVE, we complete the plan and we have a feasible fence design to orient this type of parts. A picture of the reduce is given in Figure A.11.

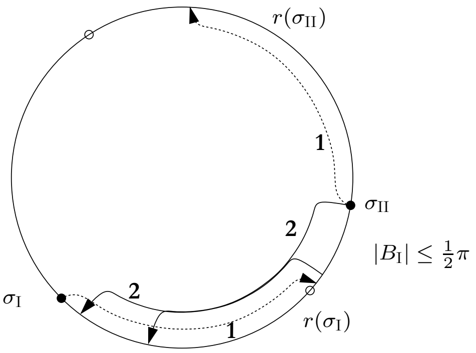
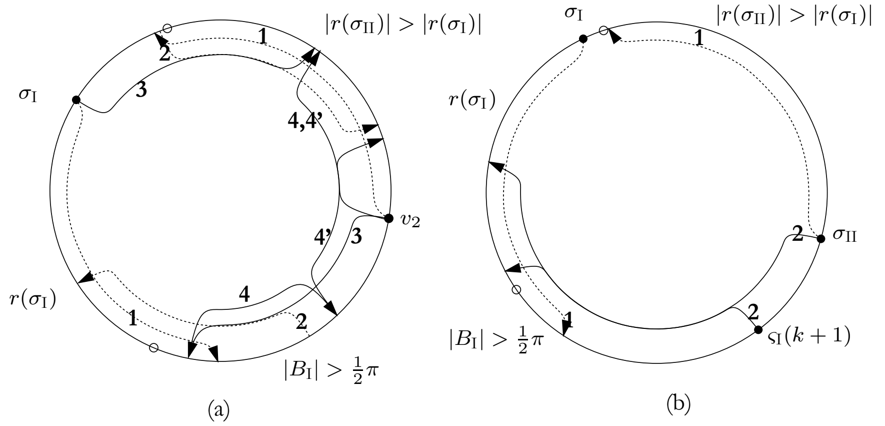


Figure A.12 The REDUCE_l for a part with two large right environment of different length. The plan is $(|r(\sigma_{\text{I}})| + \epsilon), -(\pi/2 + \epsilon)$.

Two long environments not of equal length

Let σ_{I} be the orientation with the shortest right environment greater than $\pi/2$; the other orientation is called σ_{II}, the angular interval between $r(\sigma_{\text{I}})$ and $r(\sigma_{\text{II}})$ (in counterclockwise order) is B_{I}. We distinguish between two cases for different lengths of B_{I}.

1. The first case is applicably if the length of B_{I} is less than or equal to $\pi/2$. With a reorientation of the jaw by $(|r(\sigma_{\text{I}}) + \epsilon)$ which is a push by a right fence, we get that $\varsigma_{\text{I}}(k)$ is the orientation neighboring σ_{I}, and $\varsigma_{\text{II}}(k+1) = \sigma_{\text{II}}$. With a left fence we construct a reorientation of the jaw of $-(\pi/2 + \epsilon)$, and shift $\varsigma_{\text{II}}(k+1)$ to the left, such that $\varsigma_{\text{II}}(k+2) = \sigma_{\text{I}}$ (skipping B_{I}). The other possible orientation of P, $\varsigma_{\text{I}}(k+1)$, is also mapped onto σ_{I}. Hence, $\sigma_{\text{I}} = \varsigma_{\text{I}}(k+2) = \varsigma_{\text{II}}(k+2)$. See Figure A.12 for a picture of this reduce.
2. The second case is applicably if the length of B_{I} is larger than $\pi/2$. We now know that both large right environments are smaller than π. We use a right fence to construct a reorientation of the jaw by $(|r(\sigma_{\text{II}})| + \mu)$, followed by a push, and skip both right environments. If the distance from $\varsigma_{\text{I}}(k+1)$ to the right environment of σ_{I} is less than $\pi/2$, i.e. $\varsigma_{\text{I}}(k+1) - (\sigma_{\text{I}} + |r(\sigma_{\text{I}})|) < \pi/2$, then, with again a right fence, we construct push angle $-(\pi/2 - \mu)$. Now $\varsigma_{\text{I}}(k+2) = \sigma_{\text{I}}$, and $\varsigma_{\text{II}}(k+2) = \varsigma_{\text{II}}$. In this case, we can use a left fence to push $-(\pi/2 + \mu)$ to get $\varsigma_{\text{I}}(k+3) = \sigma_{\text{II}}$, and $\varsigma_{\text{II}}(k+3) \in B_{\text{I}}$. With a MOVE, we get $\varsigma_{\text{II}}(k + 3 + |\text{MOVE}|) = \sigma_{\text{II}}$ as well.
   If $\varsigma_{\text{I}}(k+1) - (\sigma_{\text{I}} + |r(\sigma_{\text{I}})|) \geq$, we use a different plan. The $k+1$-th a push is now $|r(\sigma_{\text{II}})| - \mu$, instead. Still, $\varsigma_{\text{I}}(k+1)$ is the same orientation as it would



(a) (b)

Figure A.13 The REDUCE$_l$ plans for parts with two long right environments, and the angular interval between the shortest and the longest intervals longer than $\pi/2$. (a) $(|r(\sigma_{II})| + \epsilon), -(\pi/2 - \epsilon), -(\pi/2 + \epsilon)$. The plan end with MOVE (**4**). (b) If $-(\pi/2 - \epsilon)$ of the reduce in (a) does not map ς_I back onto σ_I. The plan is different: $(|r(\sigma_{II})| - \epsilon), -(|B_I| + \epsilon)$.

have been in the plan we described above; $\varsigma_{II}(k+1) = \sigma_{II}$ in this case, though. We now use a reorientation of the jaw of $-(|B_I| + \epsilon)$, by a left fence to achieve $\varsigma_I(k+2) = \varsigma_{II}(k+2) = \sigma_{II}$. This is possible, since $\varsigma_I(k+1) - \varsigma_{II}(k+1) < \pi/2$. See Figures A.13(a) and (b) for a picture of the corresponding push plans.

In this section we derived the following:

Theorem A.15 *Let P be a part. If P's push functions stable equilibria Σ have right cycle $|\Sigma|$ and there are right environments longer than or as long as $\pi/2$ then there is there is a fence design that orients the part up to symmetry. Similarly, if the stable equilibria Σ have left cycle $|\Sigma|$ and there are left environments longer than or as long as $\pi/2$ then there is there is a fence design that orients the part up to symmetry.*

This section completes the discussion of the different special cases for orienting a polygonal part by a fence design.

References

[1] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *Journal of Algorithms*, 17:292–318, 1994.

[2] S. Akella, W. Huang, K. M. Lynch, and M. T. Mason. Parts feeding on a conveyor with a one joint robot. *Algorithmica*, 26:313–344, 2000.

[3] S. Akella and M. T. Mason. Posing polygonal objects in the plane by pushing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2255–2262, 1992.

[4] M. Albertson, R. Haas, and J. O'Rourke. Some results on clamping a polygon. *Congressus Numerantium*, 109:33–50, 1995.

[5] G. Barequet. DCEL: A polyhedral database and programming environment. *International Journal of Computational Geometry and Applications*, 8:619–636, 1998.

[6] S. Basu. *Algorithms in Semi-algebraic Geometry*. Ph.D. thesis, Department of Computer Science, New York University, 1996.

[7] S. Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *Journal of the A.C.M.*, 46(4):537–555, 1999.

[8] S. Basu, R. Pollack, and M-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the A.C.M.*, 43:1002–1045, 1996.

[9] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink, and S. Yu. The complexity of rivers in triangulated terrains. In *Canadian Conference on Computational Geometry*, pages 325–330, 1996.

[10] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[11] D. Berkowitz and J. Canny. Designing parts feeders using dynamic simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1127–1132, 1996.

[12] R-P. Berretty, K. Y. Goldberg, L. Cheung, M. H. Overmars, and A. F. van der G. Smith Stappen. Trap design for vibratory bowl feeders. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2558–2563, 1999.

[13] R-P. Berretty, K. Y. Goldberg, M. H. Overmars, and A. F. van der Stappen. On fence design and the complexity of push plan for orienting parts. In *Annual A.C.M. Symposium on Computational Geometry*, pages 21–29, 1997.

[14] R-P. Berretty, K. Y. Goldberg, M. H. Overmars, and A. F. van der Stappen. Algorithms for fence design. In *Robotics, the algorithmic perspective*, pages 279–295. A.K. Peters, 1998.

[15] R-P. Berretty, K. Y. Goldberg, M. H. Overmars, and A. F. van der Stappen. Computing fence designs for orienting parts. *Computational Geometry: Theory and Applications*, 10(4):249–262, 1998.

[16] R-P. Berretty, K. Y. Goldberg, M. H. Overmars, and A. F. van der Stappen. Geometric techniques for trap design. In *Annual A.C.M. Symposium on Computational Geometry*, pages 95–104, 1999.

[17] R-P. Berretty, K. Y. Goldberg, M. H. Overmars, and A. F. van der Stappen. Geometric trap design for automatic part feeders. In *International Symposium on Robotics Research*, pages 139–144, 1999.

[18] R-P. Berretty, M. H. Overmars, and A. F. van der Stappen. Orienting polyhedral parts by pushing. In *Proceedings of Euro-CG*, pages 136–139, 2000.

[19] R-P. Berretty, M. H. Overmars, and A. F. van der Stappen. Orienting polyhedral parts by pushing. Technical report, UU-CS-2000-21, De-

partment of Computer Science, Utrecht University, 2000. submitted to Computational Geometry, Theory and Applications.

[20] K-F. Böhringer, V. Bhatt, B.R. Donald, and K. Y. Goldberg. Algorithms for sensorless manipulation using a vibrating surface. *Algorithmica*, 26:389–429, 2000.

[21] G. Boothroyd and P. Dewhurst. *Design for Assembly – A Designers Handbook*. Department of Mechanical Engineering, University of Massachusetts, Amherst, Mass., 1983.

[22] G. Boothroyd, C. Poli, and L. Murch. *Automatic Assembly*. Marcel Dekker, Inc., New York, 1982.

[23] P. Bose, D. Bremner, and G. Toussaint. All convex polyhedra can be clamped with parallel jaw grippers. *Computational Geometry: Theory and Applications*, 6:291–302, 1996.

[24] W. Boyes, editor. *Handbook of Jig and Fixture Design, 2nd Edition*. Society of Manufacturing Engineers, 1989.

[25] M. Brokowski, M. A. Peshkin, and K. Y. Goldberg. Optimal curved fences for part alignment on a belt. *ASME Transactions of Mechanical Design*, 117, 1995.

[26] R. Brost. Automatic grasp planning in presence of uncertainty. *International Journal of Robotics Research*, 7(1):3–17, 1988.

[27] R. C. Brost and K. Y. Goldberg. A complete algorithm for synthesizing modular fixtures for polygonal parts. *IEEE Transactions on Robotics and Automation*, 12:31–46, 1996.

[28] M. E. Caine. The design of shape interaction using motion constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 366–371, 1994.

[29] J. Canny and K. Y. Goldberg. Risc for industrial robotics: Recent results and open problems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1951–1958, 1994.

[30] B. Chazelle. Computational geometry: A retrospective. In *Annual A.C.M. Symposium on Theory of Computing*, pages 75–94, 1994.

[31] J. Chen, K. Y. Goldberg, M. H. Overmars, D. Halperin, K-F. Böhringer, and Y. Zhuang. Shape tolerance in feeding and fixturing. In *Robotics, the*

algorithmic perspective, pages 297–311. A.K. Peters, 1998.

[32] Y-B. Chen and D. J. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algoritmica*, 14:367–397, 1995.

[33] L. P. Chew and K. Kedem. Placing the largest similar copy of a convex polygon among polygonal obstacles. In *Annual A.C.M. Symposium on Computational Geometry*, pages 167–174, 1989.

[34] Y-C. Chou, V. Chandry, and M. M. Barash. A mathematical approach to automatic configuration of machining fixtures: Analysis and synthesis. *Journal of Engineering for Industry, Transactions of the ASME*, 111:199–306, 1990.

[35] A. Christiansen, A. Edwards, and C. Coello. Automated design of parts feeders using a genetic algorithm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–851, 1996.

[36] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *GI Conference on Automata Theory and Formal Languages, LNCS*, volume 33, pages 134–183, 1975.

[37] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts London, England, 1990.

[38] C. A. Coulomb. Théorie des machines simples en ayant égard au frottement de leurs parties et à la roideur des cordages. In *Mémoires des mathématique et de physique présentés à l'Académie des Sciences*, 1781.

[39] L. S. Homem de Mello and S. Lee, editors. *Computer Aided Mechanical Assembly*. Kluwer Academic Publishers, Boston, 1991.

[40] D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal of Computing*, 19(5):500–510, 1990.

[41] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4:367–379, 1988.

[42] M. A. Erdmann, M. T. Mason, and G. Vaněček. Mechanical parts orienting: The case of a polyhedron on a table. *Algorithmica*, 10(2):226–247, 1993.

[43] C. Ferrari and J. Canny. Planning optimal grasps. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 12, pages 31–46, 1996.

[44] G-J. Giezeman. PlaGeo - a library for planar geometry. Technical report, Department of Computer Science, Utrecht University, Utrecht, Netherlands, 1993.

[45] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algoritmica*, 10(2):201–225, 1993.

[46] K. Y. Goldberg. Completeness in robot motion planning. In *The First Workshop on Algorithmic Foundations of Robotics*, pages 419–430, Boston, MA, 1994. A.K. Peters.

[47] K. Y. Goldberg, B. Mirtich, Y. Zhuang, J. Craig, B. Carlisle, and J. Canny. Part pose statistics: Estimators and experiments. *IEEE Transactions on Robotics and Automation*, pages 849–859, 1999.

[48] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, 1997.

[49] D. Halperin, J-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.

[50] J. Heintz, T. Recio, and M-F. Roy. Algorithms in real algebraic geometry and applications to computational geometry. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 6:137–163, 1991.

[51] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33:169–174, 1989.

[52] W. Huang and M. T. Mason. Mechanics, planning and control for tapping. In *Robotics, the algorithmic perspective*, pages 91–102. A.K. Peters, 1998.

[53] M. Jakiela and J. Krishnasamy. Computer simulation of vibratory parts feeding and assembly. In *International Conference on Discrete Element Methods*, pages 403–411, 1993.

[54] L. E. Kavraki and M. N. Kolountzakis. Partitioning a planar assembly into two connected components is NP-complete. *Information Processing Letters*, 55(3):159–165, 1995.

[55] D. Kriegman. Let them fall where they may: Capture regions of curved objects and polyhedra. *International Journal of Robotics Research*, 16:448–472, 1997.

[56] J-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers,

Boston, 1991.

[57] D. Leven and M. Sharir. On the number of critical free contacts of a convex polygonal object moving in two-dimensional polygonal space. *Discrete & Computational Geometry*, 2:255–270, 1987.

[58] L. Lim, B. Ngoi, S. Lee, S. Lye, and P. Tan. A computer-aided framework for the selection and sequencing of orientating devices for the vibratory bowl feeder. *International Journal of Production Research*, 32(11):2513–2524, 1994.

[59] K. M. Lynch. Inexpensive conveyor-based parts feeding. *Assembly Automation Journal*, 19(3):209–215, 1999.

[60] K. M. Lynch. Locally controllable manipulation by stable pushing. *IEEE Transactions on Robotics and Automation*, pages 314–323, 1999.

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

[62] A. Marigo, M. Ceccarelli, S. Piccinocchi, and A. Bicchi. Planning motions of polyhedral parts by rolling. *Algorithmica*, 26(4):560–576, 2000.

[63] X. Markenscoff, L. Ni, and C. H. Papadimitriou. The geometry of grasping. *International Journal of Robotics Research*, 9(1):61–74, 1990.

[64] X. Markenscoff and C. H. Papadimitriou. Optimal grip of a polygon. *International Journal of Robotics Research*, 8(2):17–29, 1989.

[65] M. T. Mason. Mechanics of robotic manipulation. unpublished book.

[66] M. T. Mason. *Manipulator grasping and pushing operations*. PhD thesis, MIT, 1982. published in *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, 1985.

[67] M. T. Mason. Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research*, 5(3):53–71, 1986.

[68] M. T. Mason. Two graphical methods for planar contact problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1991.

[69] G. Maul and M. Thomas. A systems model and simulation of the vibratory bowl feeder. *Journal of Manufacturing Systems*, 16(5):309–314, 1997.

[70] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the A.C.M.*, 30(4):852–865, 1983.

[71] B. Mishra. *Algorithmic Algebra.* Springer-Verlag, New York Berlin Heidelberg, 1993.

[72] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

[73] P. Moncevicz, M. Jakiela, and K. Ulrich. Orientation and insertion of randomly presented parts using vibratory agitation. In *ASME Conference on Flexible Assembly Systems*, pages 41–47, 1991.

[74] H. Moseley. *Illustrations of Mechanics.* London, 1938.

[75] B. K. Natarajan. On planning assemblies. In *Annual A.C.M. Symposium on Computational Geometry*, pages 299–308, 1988.

[76] B. K. Natarajan. Some paradigms for the automated design of parts feeders. *International Journal of Robotics Research*, 8(6):89–109, 1989.

[77] V-D. Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3):3–16, 1988.

[78] M. H. Overmars, A. Rao, O. Schwarzkopf, and C. Wentink. Immobilizing polygons against a wall. In *Annual A.C.M. Symposium on Computational Geometry*, pages 29–38, 1995.

[79] M. A. Peshkin and A. C. Sanderson. The motion of a pushed sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6):569–598, 1988.

[80] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, pages 696–701, 1988.

[81] J. Ponce and B. Faverjon. On computing three-finger force-closure grasps of polygonal objects. *IEEE Transactions on Robotics and Automation*, 11(6):868–881, 1995.

[82] J. Ponce, D. Stam, and B. Faverjon. On computing force-closure grasps of curved two-dimensional objects. *International Journal of Robotics Research*, 12:263–273, 1993.

[83] J. Ponce, S. Sullivan, A. Sudsang, J-D. Boissonnat, and J-P. Merlet. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *International Journal of Robotics Research*, 16(1):13–35, 1997.

[84] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.*

Springer-Verlag, 3rd edition, 1990.

[85] A. Rao and K. Y. Goldberg. Shape from diameter: Recognizing polygonal parts with a parallel-jaw gripper. *International Journal of Robotics Research*, 13(1):16–37, 1994.

[86] A. Rao and K. Y. Goldberg. Friction and part curvature in parallel-jaw grasping. *Journal of Robotic Systems*, 12(6):365–382, 1995.

[87] A. Rao and K. Y. Goldberg. Manipulating algebraic parts in the plane. *IEEE Transactions on Robotics and Automation*, 11:589–602, 1995.

[88] A. Rao, D. Kriegman, and K. Y. Goldberg. Complete algorithms for reorienting polyhedral parts using a pivoting gripper. *IEEE Transactions on Robotics and Automation*, 12(2):331–342, 1996.

[89] F. Reuleaux. *The Kinematics of Machinery*. Macmilly and Company, 1876. Republished by Dover in 1963.

[90] D. Reznik and J. Canny. Universal part manipulation in the plane with a single horizontally-vibrating plate. In *Robotics, the algorithmic perspective*, pages 23–34. A.K. Peters, 1998.

[91] E. Rimon and J. W. Burdick. Mobility of bodies in contact—part I: A second-order mobility index for multiple-finger graps. *IEEE Transactions on Robotics and Automation*, 14:696–708, 1998.

[92] E. Rimon and J. W. Burdick. Mobility of bodies in contact—part II: How forces are generated by curvature effects. *IEEE Transactions on Robotics and Automation*, 14:709–717, 1998.

[93] S. Rusaw, K. Gupta, and S. Payandeh. Orienting polygons with fences over a conveyor belt: Empirical results. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 831–836, 1998.

[94] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:289–351, 1983.

[95] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[96] M. Sharir and S. Toledo. Extremal polygon containment problems. *Computational Geometry: Theory and Applications*, 4:99–118, 1994.

[97] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. *Discrete & Computational Geometry*, 12:367–384, 1994.

[98] Sony. Advanced parts orienting system (apos). Technical Report 801-8902-11, Sony Corporation, 1989.

[99] D. Souvaine and C. van Wijk. Clamping a polygon. *The visual computer*, pages 484–494, 1994.

[100] A. F. van der Stappen, K. Y. Goldberg, and M. H. Overmars. Geometric eccentricity and the complexity of manipulation plans. *Algorithmica*, 26:494–514, 2000.

[101] A. F. van der Stappen, C. Wentink, and M. H. Overmars. Computing immobilizing grasps of polygonal parts. *International Journal of Robotics Research*, 19(5):467–479, 2000.

[102] A. Sudsang, J. Ponce, and N. Srinivasa. Algorithms for constructing immobilizing fixtures and grasps of three-dimensional objects. In J-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 363–380. A.K. Peters, 1997.

[103] S. Toledo. Extremal polygon containment problems and other issues in parametric searching. M.s. thesis, Department of Computer Science, Tel Aviv University, Tel Aviv, 1991.

[104] L. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[105] R. Wagner, Y. Zhuang, and K. Y. Goldberg. Fixturing parts with seven modular struts. In *IEEE International Symposium on Assembly and Task Planning*, pages 133–139, 1995.

[106] A. S. Wallack and J. Canny. Planning for modular and hybrid fixtures. *Algorithmica*, 19(1–2):40–60, 1997.

[107] C. Wentink. *Fixture Planning—Geometry and Algorithms*. PhD thesis, Department of Computer Science, Utrecht University, 1998.

[108] C. Wentink, A. F. van der Stappen, and M. H. Overmars. Algorithms for fixture design. In J-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 321–346. A.K. Peters, 1997.

[109] D. E. Whitney. Real robots don't need jigs. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 746–752, 1986.

[110] J. A. Wiegley. *Sorting convex polygonal parts without sensors of a conveyor.* PhD thesis, Faculty of the graduate school, University of Southern California, 1998.

[111] J. A. Wiegley, K. Y. Goldberg, M. Peshkin, and M. Brokowski. A complete algorithm for designing passive fences to orient parts. *Assembly Automation*, 17(2):129–136, 1997.

[112] R. H. Wilson. *On Geometric Assembly Planning.* PhD thesis, Stanford University, 1992. Stanford Technical Report STAN-CS-92-1416.

[113] R. H. Wilson and J-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71:371–396, 1994.

[114] R. Zhang and K. Gupta. Automatic orienting of polyhedra through step devices. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 550–556, 1998.

[115] T. Zhang, G. Smith, R-P. Berretty, M. H. Overmars, and K. Y. Goldberg. The toppling graph: Designing pin sequences for part feeding. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 139–146, 2000.

[116] Y. Zhuang and K. Y. Goldberg. On the existence of solutions in modular fixturing. *International Journal of Robotics Research*, 15:646–656, 1996.

Symbols

Lower case roman

c	center-of-mass
d	period, 18
e	edge
f	fence, or force
h	height
i, j, $\bar{\imath}$, $\bar{\jmath}$, $\tilde{\imath}$, $\tilde{\jmath}$	integers (usually indices)
k	size of output
$l(\cdot)$	left environment, 17
m, n	cardinality of a set
$r(\cdot)$	left environment, 17
t	fence type
v	vertex
x, y, z	reals

Upper case roman

H	half plane
I	interval
P	part
Q	quantifier, i.e. $\forall$, or $\exists$

S	supported area of trap
T	trap

Script

$\mathcal{I}$	interval of possible reorientations
$\mathcal{C}$	circle
$\mathcal{S}$	semi-algebraic set
$\mathcal{T}$	tree

Lower case greek

α	angle
δ, $\delta(\cdot)$	distance, radius function, 16
ϵ	positive real, usually small
γ	minimum length of left and right environments, 22
λ, $\reflectbox{$\lambda$}$	length of the longest right, or left environment
μ	coefficient of friction
ϕ, ψ, θ	angles
ρ	ray
σ	(stable) orientation
ϖ	push, or transfer function, 17
ς	possible orientation (at some stage of a plan)
σ^*	unique (final) orientation

Upper case greek

Δ	triangle
Λ	defining formula
Σ	set of orientations

Specific to fence design

Σ^{λ}, $\Sigma^{\reflectbox{$\lambda$}}$	set of orientations with long right, and left environments, 41
$M_{\Sigma^{\lambda}}$, $M_{\Sigma^{\reflectbox{$\lambda$}}}$	right, and left measure, 41

B	sequence of orientations between orientations with large right environments
c_l, c_r	left and right cycle of the push function

Specific to inside out pulling

$\delta(\cdot)$	distance function
$\tau_v(\cdot)$	boundary intersection function from vertex v
$\varpi_v(\cdot)$	pull function for vertex v
u	unstable contact point
v	stable vertex
v^*	unique (final) stable vertex

Specific to bowl feeder design

μ, ν, γ	parameters of trap shape
τ	general trap shape
ξ	number of surfaces
d	degree of polynomial
e_s, e_c, e_l, e_u	starting, closing, lower and upper edge of trap
q	placement, or position of the trap

Miscellaneous

$\angle$	angle
ℓ	line
$\wp$	polynomial
$\mathcal{CH}$	convex hull

Index

Ackermann function, 114
action, *see* basic action
actuator, 2
acyclic, 41, 139
algebraic, 117
alignment
    fence, 25
    finger, 59
    jaw, 14
    primary plane, 77
    secondary plane, 81
antipodal, 82
application
    jaw, 18
    primary plane, 77
    secondary plane, 81
arbitrary
    part, 40, 68
    trap, 116
areas
    of rotation, 14, 44
arrangement, 87, 97, 116, 117
assembly, 1
asymmetric, 39, 40, 84
    polyhedral part, 76

balcony, 104
    design, 104
ball, 78
basic action, 5, 18
    jaw, 76
    push, 18, 20
bijection, 85
border, 113
bowl feeder, 4, 93

candidates, 34
canonical subset, 36
canyon, 104
    design, 110
cell, 86, 117, 118
center-of-mass, 14, 77
center-of-rotation, 15
Chen and Ierardi, 20, 40
circle, 61
    great, 83
coefficient of friction, 14, 44
COLLIDEPRIMARYACTION, 84

COLLIDEROLLSSEQUENCE, 82
completeness, 6, 39, 40, 66, 81
complexity
    radius terrain, 85
computational geometry, 1
configuration, 57
constraints
    see fence design, constraints, 25
contact direction, 16, 28, 77
convex, 17
    hull, 17, 96
    part, 17, 23, 26, 56, 100
    trap, 100
conveyor belt, 2, 25
coset, 131
Coulomb
    friction, 12
critical
    placement, 96
    shape, 96
cycle, 41
    left, 41
    right, 41
cyclic, 41, 133
cylinder, 74, 89

Davenport-Schinzel sequence, 114
degenerate, 79
degrees of freedom, 74, 103
device, 15, 56, 73, 93
diagonal, 77, 78
disc, 77
distance
    function, 57
doubly-connected edgelist, 86

eccentricity, 52
edge
    contact, 14
envelope
    left tangent, 112
    upper, 108
    upper tangent, 109
environment, 18
    left, 18
    long, 40
    right, 18
Eppstein, 19, 25, 69
equator, 83
equilibrium, 17, 18
    configuration, 57
    contact point, 57
    direction, 17
    orientation, 17, 18, 76
    stable, 18, 29
    stable pose, 17
    stable vertex, 57
    unstable, 14, 91
Euclidean, 117
event, 99
exponential algorithm, 7, 25, 119
extremal rays, 98

fed, 96
feeder, 4, 18, 32, 56, 73, 93
feeding property, 18, 19, 56, 96
fence, 25, 27, 89
    adjustable, 47
    curved, 27
    left, 27
    right, 27
fence design, 25, 28
    algorithm, 29
    alternating, 51
    completeness, 39, 40, 43
    computing, 29
    constraints, 25, 28
    friction, 44

    friction algorithm, 45
    friction constraints, 45
    friction model, 44
    model, 27
    modular, 26, 47
    modular algorithm, 49
    modular model, 47
    output-sensitive algorithm, 33
    three-dimensional, 74
filter, 93
fixture planning, 3
force
    frictional, 12
    normal, 12
friction, 12
    coefficient of, 12
    cone, 12, 44
    Coulomb, 12
    fence design, 44

gap, 104
    design, 106
Goldberg, 20, 70
graph, 29, 30, 67
    reduced, 32
grasping, 3, 55
gravity, 12, 89
greedy algorithm, 22

half-step, 18
hemisphere, 83

implementation, 51, 87, 94
inertia, 11, 76
inside out
    grasping, 55
    pulling, *see* pull
interval, 17, 34
    of orientations, 21, 29

maintain
    intervals, 33
manipulator, 16
manufactoring, 2
measure, 41
mechanics, 11
meta-stable, 23
modular
    fences, 47
monotonic, 18, 61
    pull function, 61
    push function, 20
motion, 12
    belt, 25
    part, 12, 13
MOVE, 41

normal, 12, 25, 77
NP-complete, 2, 69, 92

ordered, 18, 65
    set, 18
orienting, 4
    sensorless, 4
output-sensitive, 33, 68

parallel jaw gripper, 16, 55
parameters
    trap, 103, 116
part
    feeding, 4
    shape, 16
path, 30
period, 18
peshkin bound, 15, 27
placement, 95
plageo, 51
planar projection, 85

plane
  primary, 75
  secondary, 75
plate, 74, 89
polygonal part, 16, 22
polyhedral part, 74
pressure distribution, 13
projection, 23, 76
pull, 55
  action, 59
  algorithm, 67
  basic action, 59
  completeness, 66
  computing, 67
  function, 59
  model, 56
  output sensitive, 68
  plan, 56, 66
pulling, *see* pull
push, 5, 11
  by fence, 25
  by generic jaw, 76
  by plates and fences, 75
  by primary plane, 77
  direction, 44
  function, 16, 20, 22, 76
  plan, 20, 28, 74
  planar function, 81
  secondary plane, 81
  three-dimensional model, 75
pushing, *see* push
  jaw, 13

quasi-static, 11, 76, 89

radius, 16, 77
  function, 16, 77, 95
  terrain, 78
range tree, 35
  query, 36
REDUCE, 41
reduced
  graph, 32
reject, 96
reorientation, 133
  jaw, 28
  push direction, 28, 45
representatives, 116
RISC, 2, 7, 9
robot, 1
robotic manipulation, 1
roll
  computation, 79
rotate, 14
rotation
  mode of, 13
  simple, 79

saddle point, 78
safe, 95
sample point, 116
semi-algebraic, 117
sensorless
  orienting, 4
shape, 96
SHIFT, 41
sliding, 13, 25
slot, 104
  design, 114
solution-complete, 6
sphere of directions, 74, 83
square, 121
squeeze, 5, 55
stable, 18, 29
  orientation, 81
state, 29, 67
steepest descent, 78, 79, 85
step, 17

  function, 17
step function, 60
Stretching Lemma, 21, 40, 67
sweep line
  algorithm, 99
symmetry, 18, 38, 76
  periodic, 18

Tarski set, 117
three-dimensional, 74
  orientation, 74
tip point, 15
toppling, 6
track, 93
transfer, 5
  function, 18
trap, 5, 93
  analysis, 96, 97, 100
  arbitrary, 116
  design, 103
  rectangular, 103
tray, 5

uncertainty, 53, 121
unsafe, 95
unstable, *see* equilibrium

vertex
  contact, 14
vibratory
  bowl feeder, *see* bowl feeder

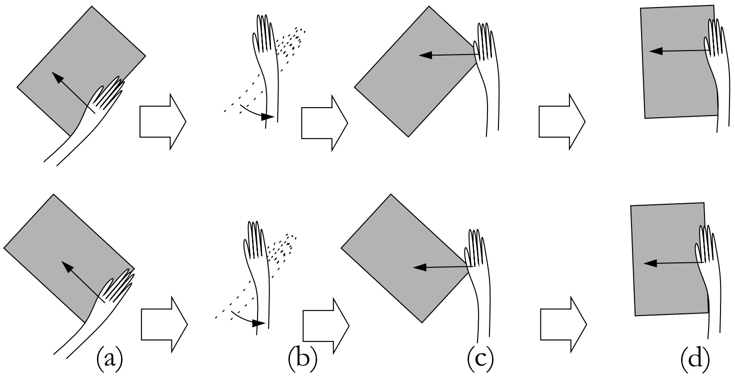
wobbling, 5
workspace, 1
wrench, 3

Samenvatting

Tegenwoordig worden veel processen met behulp van computers of computergestuurde robots uitgevoerd om efficiënter te kunnen produceren. Bij vroege pogingen om met robots menselijke taken uit te voeren streefde men ernaar robots te maken die haast zuivere kopieën van de mens waren. Die eerste robots werden ontwikkeld vanuit de gedachte dat ze op een menselijke manier zouden moeten kunnen omgaan met problemen op de fabrieksvloer. Pas later realiseerde men zich dat het efficiënter zou zijn robots te ontwikkelen uitgaande van de *taak* die deze moesten kunnen uitvoeren. Niet de mens, maar het werk dat verricht moest worden werd de maatstaf voor de ontwikkeling van computergestuurde systemen.

Dit proefschrift gaat over taakgerichte oplossingen voor problemen die zich in de industriële assemblage voordoen bij het inzetten van robots. Belangrijke overweging bij het ontwikkelen van een geautomatiseerd systeem zoals de robot, is dat de betrouwbaarheid ervan groter wordt naarmate er minder bewegende delen, sensoren of camera's gebruikt worden. Bij het ontwikkelen van taakgerichte robots is het wenselijk de machine die de taak gaat uitvoeren zo eenvoudig mogelijk te construeren, zonder dat daarbij aan functionaliteit wordt ingeboet.

We onderzoeken met het oog op het vinden van taakgerichte oplossingen apparaten die steeds op uniforme wijze onderdelen aan robots aanleveren. Zo'n apparaat—dat onderdelen in een vaste *oriëntatie* (onder een vaste hoek) aan de robot aanlevert—wordt een *onderdeelvoeder* (Eng: *part feeder*) genoemd. Als het onderdeel eenmaal georiënteerd is (dat wil zeggen: in de juiste stand is aangereikt), kan een robot het vervolgens op snelle en efficiënte wijze oppakken en verder verwerken. Op het eerste gezicht lijkt het oplossen van problemen in

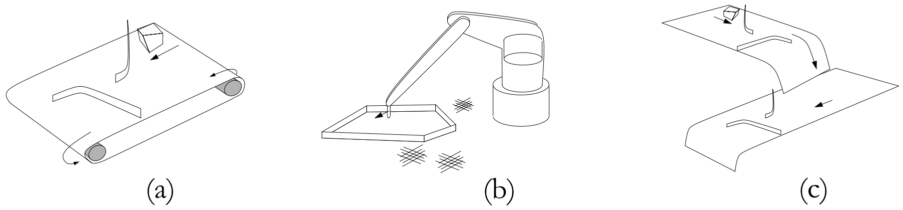


Figuur 1 Oriënteren met behulp van duwacties.

dit proces gezocht te moeten worden in de manier waarop onderdelen aan de robot aangereikt, en dus georiënteerd worden. Problemen tijdens de assemblage lijken alleen opgelost te kunnen worden als de onderdeelvoeder óf de initiële oriëntatie van het onderdeel kent, óf die eerst vaststelt (bijvoorbeeld aan de hand van een camerabeeld). De gedachte is dat wanneer de initiële oriëntatie bekend is, met een simpele manipulatie van het onderdeel de gewenste eindoriëntatie bereikt worden. Uit eerder onderzoek is echter gebleken dat voor het zoeken van een unieke eindoriëntatie niet altijd kennis over de beginoriëntatie nodig is.

Onderzoek laat zien dat slechts de vorm van het onderdeel bekend hoeft te zijn om tot een geslaagde eindoriëntatie te komen. We concentreren ons in dit onderzoek op onderdeelvoeders die eenvoudig en flexibel zijn. Veel van de oplossingsmethoden gericht op het oriënteren van onderdelen zonder het gebruik van sensoren, gaan uit van series van eenvoudige acties. Die acties zorgen ervoor dat het onderdeel steeds een klein beetje verdraaid wordt. Een voorbeeld van zo'n eenvoudige actie is de duwbeweging. Door tegen een object dat op de tafel ligt aan te duwen, kunnen onzekerheden in de oriëntatie en positie van het object worden geëlimineerd.

De lezer kan bij wijze van experiment en toets dit boek op tafel leggen en er met de vlakke hand aan de zijkant tegenaan duwen. Zie Figuur 1. Afhankelijk van de duwrichting zal het boek uiteindelijk met de korte of met de lange zijde tegen de hand aanliggen (Figuur 1(a)). Als we nu de hand weer wegnemen van het boek en opnieuw gaan duwen (Figuur 1(b)), vanuit een richting die onder een hoek van 45 graden staat met de eerste duwrichting (Figuur 1(c)), zal het boek altijd met de lange zijde in contact komen met de


(a) (b) (c)

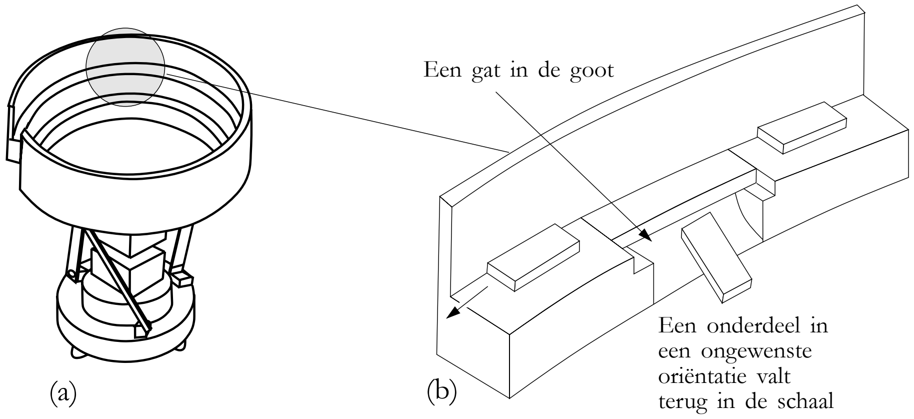
Figuur 2 Drie van de voeders die aan de orde komen in dit onderzoek. (a) Hekjes boven een lopende band. (b) De trekkende vinger. (c) Platen met hekjes.

hand (Figuur 1(d)). Met twee duwacties hebben we er dus voor gezorgd dat de onzekerheid in de oriëntatie van het boek is geëlimineerd.

Eerder onderzoek heeft uitgewezen dat het mogelijk is om *elk* object dat op de tafel ligt te oriënteren, tenzij het duwgedrag van het onderdeel symmetrisch is—het is bijvoorbeeld niet mogelijk om onderscheid te maken tussen de vier zijden van een vierkant.

In dit onderzoek is naar vier soorten voeders gekeken, en gezocht naar vier series van eenvoudige acties die het voedprobleem van de assemblage oplossen. De eerste onderzochte voeder werkt grotendeels door duwacties. Hekjes, achter elkaar op een lopende band geplaatst, zorgen in dit geval voor verandering in oriëntatie (zie Figuur 2(a)). Het oriënteren van onderdelen met behulp van deze hekjes was eerder onderwerp van onderzoek, maar op basis daarvan konden geen harde uitspraken worden gedaan over de mogelijkheid om elk onderdeel te kunnen oriënteren door de duwkrachten van de hekjes. In dit onderzoek wordt voor het eerst aangetoond dat elk onderdeel zonder symmetrisch duwgedrag georiënteerd kan worden met behulp van hekjes boven een lopende band. We geven daarnaast efficiënte methoden om de stand van de hekjes te berekenen aan de hand van de specificatie van het te oriënteren onderdeel.

De tweede voeder die we in dit onderzoek bekijken, werkt op basis van trekacties. We analyseren een robotvinger, gepositioneerd boven een onderdeel met een opstaande rand (zie Figuur 2(b)). De vinger beweegt parallel aan het werkblad en trekt daar zo het onderdeel aan de opstaande rand overheen. Tijdens de trekbeweging verandert de oriëntatie van het onderdeel. We laten zien dat er voor een belangrijke klasse van onderdelen altijd een serie trekoperaties gevonden kan worden die resulteert in een vaste eindoriëntatie. Ook laten we zien dat er bepaalde onderdelen zijn die niet te oriënteren zijn. We geven berekeningsmethoden die aan de hand van de vorm van onderdelen series van trekoperaties berekenen om ze te oriënteren.



Figuur 3 (a) Een trilvoeder. (b) De werking van een gat in de goot. Het onderdeel in een ongewenste oriëntatie valt terug in de schaal.

Voor de derde voeder zijn bevindingen uit eerdere theoretische studies naar de mogelijkheden van onderdeelvoeders van belang. Veel daarvan hebben zich beperkt tot het oriënteren van onderdelen op een werkblad. Men nam daarbij aan dat bekend is welk zijvlak van het onderdeel in contact is met het werkblad, of dat het bestudeerde onderdeel plat is. Technisch gesproken wordt in dergelijk onderzoek geen driedimensionaal probleem beschouwd, maar slechts een tweedimensionaal deelprobleem. De eerdergenoemde hekjes boven de lopende band, en de trekkende vinger zijn voorbeelden van voeders die in het vlak werken, en dus oplossingen bieden voor tweedimensionale onderdeelvoedproblemen.
De derde onderdeelvoeder die in proefschrift aan de orde komt, is er één die het driedimensionale voedprobleem oplost. We laten zien hoe het mogelijk is een onderdeel met vlakke zijden te oriënteren door het over een aantal platen te laten glijden waarop hekjes gemonteerd zijn (zie Figuur 2(c)). Doordat de zwaartekracht het onderdeel tegen de platen en de hekjes aanduwt, kunnen we stellen dat deze voeder werkt dankzij duwacties. We bestuderen in deze opzet daarom ook in feite het duwen tegen driedimensionale objecten in een algemenere context. We beschouwen de denkbeeldige situatie waarin een onderdeel in de lucht zweeft, en onderzoeken het gedrag van dat onderdeel als er met twee loodrechte panelen vanuit een willekeurige richting tegenaan geduwd wordt. We laten zien dat het mogelijk is om voor elk onderdeel dat niet symmetrisch is een serie duwoperaties te ontwerpen die ervoor zorgt dat het onderdeel georiënteerd wordt. Het fraaie van deze denkbeeldige duwom-

geving is dat de resultaten die geboekt worden naar alle waarschijnlijkheid ook van toepassing zijn op andere onderdeelvoeders die driedimensionale objecten oriënteren door middel van duwacties.

De vierde en laatste onderdeelvoeder die we beschouwen, is een trillende schaal waarin een in spiraalvorm omhoogdraaiend (helisch) gootje is gemonteerd. De vibratie van de schaal is asymmetrisch, en daardoor worden de onderdelen achter elkaar omhoog langs de goot bewogen. Door nu op het traject, dat van onder in de schaal naar boven loopt, allerlei obstakels te plaatsen—zoals bijvoorbeeld pinnetjes of gaten—wordt ervoor gezorgd dat onderdelen in een ongewenste oriëntatie het einde van het traject niet bereiken. Deze zogenaamde trilvoeder werkt dus op basis van het wegfilteren van ongewenste oriëntaties. In Figuur 3(a) zien we een tekening van een trilvoeder, en in Figuur 3(b) wordt de werking van een gat in de goot geïllustreerd. Wanneer het onderdeel in een ongewenste oriëntatie is, valt het door het gat terug in de schaal, om opnieuw omhoog te gaan bewegen.

De trilvoeder wordt in de praktijk vaak toegepast, maar voor het bepalen van de juiste plaatsing van de pinnetjes en het uitfrezen van de gaten moet daarbij de hulp van experts worden ingeroepen. Die proberen vaak enkele ontwerpen uit, aan de hand van experimenten, om zo de plaatsing te verbeteren. Dit is een tijdrovende manier van werken, die geen gegarandeerd correct eindresultaat oplevert.

Wij concentreren ons op het maken van gaten in het gootje in de schaal. Hierbij laten we zien hoe door gebruik te maken van methoden uit de computationele geometrie de afmeting van deze gaten op een systematische wijze kan worden berekend.

Samengevat kan gesteld worden dat we er in dit onderzoek in geslaagd zijn theorieën te ontwikkelen voor het oplossen van voedproblemen van uiteenlopende aard. Bij de praktische toepassing van de resultaten van dit onderzoek zijn twee kanttekeningen te plaatsen. De eerste kanttekening betreft de oplossingsmethoden voor de voedproblemen waarvoor in dit proefschrift gekozen is. Die richten zich in hoge mate op het verkrijgen van inzicht in de (geometrische) aard van de problemen. Ze zijn om die reden theoretisch van aard. We werken in een omgeving waarin de fysische eigenschappen van het onderdeel en de manipulatietechnieken relatief eenvoudig worden gemodelleerd. Deze eenvoudige modellering van de werkelijkheid is een noodzakelijke voorwaarde om een basis van algemene technieken te ontwikkelen voor de oplossing van praktijkproblemen. Juist doordat voor deze modellering gekozen is, kan van de gepresenteerde methoden bewezen worden dat ze het gewenste resultaat opleveren.

Een tweede kanttekening betreft de noodzaak van vervolgonderzoek. Een aantal onderzoekers heeft al praktijkexperimenten gedaan met enkele van de in dit proefschrift beschreven voeders. De door hen gerapporteerde bevindingen kunnen in de toekomst worden gebruikt om het model aan te passen, met als doel theorie en praktijk dichter bij elkaar te brengen.

Acknowledgements

I would like to thank everybody who has contributed to the realization of this thesis. Special thanks go to my supervisor Mark Overmars for his guidance and support all through my stay as a PhD student at the department of computer science. I am also grateful to my co-supervisor Frank van der Stappen, whose door was always open to me and my everyday concerns. The combination of Frank's critical eye and Mark's general view of the project have contributed largely to the final appearance of my thesis. I thank Ken Goldberg and his students for the pleasant co-operation and hospitality they offered during my visits with U.C. Berkeley.
I am indebted to the members of the reading committee: Prof. dr. Jan van Leeuwen, Prof. dr. ir. Frans Groen, Prof. dr. Ken Goldberg, Prof. dr. Danny Halperin, and Prof. dr. Pankaj Agarwal for reviewing my thesis and for their useful remarks. I thank Job, Els, Lieke, Guido and many others for their voluntary and critical readings of the text of this thesis. I thank Paul Wolfs for his advice on the lay-out, and Nelleke Gerlsma for designing the cover. The idea of the moving triangle in the design came from my cousin Inez. René van Oostrum took the photographs of the triangle.
I want to express my gratitude to my colleagues for creating a stimulating work environment. The act of research consists mainly of two elements: using one's powers of conception, and creating an atmosphere in which the mind functions at its best. The thinking part tends to be a split second event, but it is important to alternate this mental exertion with a good dose of physical exercise. I am especially grateful to 'players' Michiel and Bram, for the many profitable hours of vital relief we spent playing the computer game Doom over the network. I think our lives changed dramatically when

the system administrators decided to trade the single platform on which this beloved game ran so flawlessly for a diversity of computers, rendering the fierce contestants incompatible, thus ending the carnage.

Luckily the table-tennis table turned out to be an equally excellent meeting point. My life has been enriched by the acquaintance with the various playing styles of my colleagues. It is remarkable to find that the ones who excelled at Doom were less formidable players at the game of table-tennis, and vice versa. Bram, despite his penholder grip, only triumphed over me sporadically, whilst João, easily overrun in computer combat, sent me running like a headless chicken. I am especially grateful to Paul Harrenstein, who taught me how to play a decent top-spin backhand. I would also like to thank Valérie, who nearly always let me win but sometimes, to keep up the suspense, managed to produce blazing curveballs quite unexpectedly.

I am grateful for the many friends that have supported me; too many to mention them all. My final thanks go to my parents, brothers and sister. I reckon growing up in such a lively, stimulating and therefore inspiring family life has been of considerable influence on my conduct in life. Knowing I could always rely on the havoc back home has strengthened my decision to retreat for a few years of research at an academic institution.

Curriculum Vitae

Robert-Paul Mario Berretty is 19 december 1972 geboren te Nijmegen. In mei 1991 behaalde hij het VWO diploma aan het Strabrecht College te Geldrop. In september van datzelfde jaar begon hij met de studie informatica aan de Universiteit Utrecht. Hij volgde de specialisatierichting algoritmisch ontwerpen en studeerde in mei 1996 cum laude af op het onderwerp robot motion planning.
Vanaf juli 1996 tot en met december 2000 was hij in dienst van de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (N.W.O.) en gedetacheerd aan het Instituut voor Informatica en Informatiekunde in de groep toegepaste algoritmen van Prof. dr. M.H. Overmars. Hij verrichtte daar het in dit proefschrift beschreven onderzoek.