On the Design of Guillotine Traps for Vibratory Bowl Feeders

Onno C. Goemans* Anthony Levandowski‡ Ken Goldberg‡ A. Frank van der Stappen*

*Institute of Information and Computing Sciences
Utrecht University, PO Box 80089
3508 TB Utrecht, The Netherlands

‡Department of Industrial Engineering and Operations Research
University of California at Berkeley
Berkeley, CA 94720, USA

***Abstract*— The vibratory bowl feeder remains the most common approach to the automated feeding (orienting) of industrial parts. We study the algorithmic design of a trap in the bowl feeder track that filters out all but one orientation of a given polygonal part. We propose a new class of traps that we call *guillotine traps*, which remove a portion of the track between two parallel lines. A major advantage of guillotine traps over previously studied traps [1] is that they permit feeding the part in a user-specified stable orientation, whereas these other traps offered no control over the orientation to be fed. The capability of feeding a part in any priorly specified orientation for example offers the user a means of control over the feed rate.**

We present a complete algorithm that takes as input any polygonal part consisting of n vertices, along with its center of mass, and a desired output orientation of the part. Our algorithm computes a guillotine trap for a vibratory bowl feeder that outputs parts in the desired orientation, or reports that no such trap exists. The algorithm runs in $O(n\alpha(n)\log n + nk)$, where $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function, and k is the number of candidate solutions. Although the value of k is trivially bounded by $O(n)$, we conjecture that k is a small constant except for highly symmetric and regular parts. Surprisingly, our algorithm is considerably more efficient than the algorithm for the more restricted and hence less powerful gap trap, which was shown to run in $O(n^2 \log n)$ time [1]. We have implemented our complete algorithm in Mathematica and C++.

I. Introduction

A *part feeder* takes in a stream of identical parts in arbitrary orientations and outputs them in a uniform orientation. We consider the problem of *sensorless orientation* of parts, in which the initial orientation 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 orientation into a unique final orientation. Among the sensorless part feeders considered in literature are the parallel-jaw gripper [2], [3], the single pushing jaw [4], [5], [6], the conveyor belt with a sequence of (stationary) fences placed along its sides [7], [8], [2], the conveyor belt with a single rotational fence [9], the tilting tray [10], [11], vibratory plates and programmable vector fields [12].

The oldest and still most common approach to automated feeding is the vibratory bowl feeder. It consists of a bowl filled

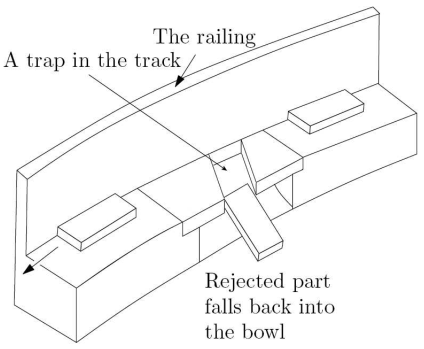

Fig. 1. Vibratory bowl feeder track with guillotine trap.

with parts surrounded by a helical metal track [13]. 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 parts emerges at the top after successfully running the gauntlet. Such a device is said to *feed* a part, if it allows the part to pass in only one specific orientation.

Feeder manufacturing still relies on skill, experience and simple guidelines [14]. Currently, the largest cost of bowl feeders is related to the design of the custom mechanisms used to feed an individual part. The costs and time associated with design of part feeders remains a barrier to flexible automation. In a typical scenario, the feeder bowl takes 200 hours to manufacture; 95 % of this time is spent designing the bowl [15]. It is evident that automation of the design process would greatly reduce the production costs and time.

In this light, researchers have used simulation [16], [17], [18], heuristics [2], [14], and genetic algorithms [19] to design traps. Geometric analysis tools have been developed that help designers visualize the configuration-space of a given combination of parts, obstacle and trap [20]. We focus on one of the most commonly used mechanisms based on removing sections of the track, the so-called traps [21], [1], [22], [23]. Agarwal et al [21] described the design of minimal traps to filter parts. Closest to our work is the research done by Berretty et al.[1]. They proposed several classes of traps

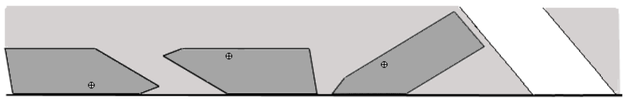

Fig. 2. A gap trap can not feed this part (hence neither of the three depicted orientations), while the guillotine trap on the right can feed the part in the third orientation.

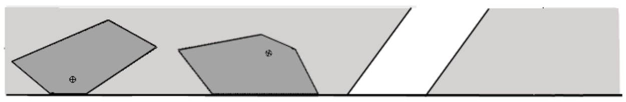

Fig. 3. The first figure depicts the orientation that can be fed by the gap trap, while the guillotine trap can feed both shown orientations. The illustrated guillotine trap feeds the second orientation. The advantage gained lies in the fact that this second orientation is estimated to occur at least twice as often, hence the usage of the guillotine trap doubles the feed rate.

as well as algorithms to compute those traps. The majority of these traps (canyon, slot, balcony, and gap) are however inherently capable of feeding only one stable orientation, which cannot be controlled by the user. This is problematic since the specific orientation can be one that occurs with extremely low probability, which would imply extremely poor feed rates. It can also be one in which the contours of the part strongly resemble its contours in another orientation. In practice, this could easily lead to false acceptances and/or rejections. In addition to the aforementioned traps, Berretty et al. introduced a more powerful class of polygonal traps. Unfortunately, their automated design is practically infeasible due to prohibitively high running time of the algorithm.

We propose a new class of traps which we call *guillotine traps*. These traps are created by removing a section of the feeder track between two parallel lines, and can be characterized by the distance g between the parallel lines, and the angle φ between the lines and the track. (Note that gap traps are a subset of guillotine traps with $\varphi = \pi/2$). Guillotine traps offer two important advantages over existing traps. Instead of dictating the orientation to be fed, they allow the designer to specify a desired stable orientation, and thus providing the the designer with a trap more powerful (Fig. 2) and flexible (Fig. 3) than any individual other trap. Moreover, even though guillotine traps generalize gaps, the algorithm for designing guillotine traps is considerably more efficient than algorithm for the less powerful gap.

The running time of our algorithm for computing guillotine traps is $O(n\alpha(n)\log n + nk)$, where $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function and k is the number of candidate solutions. The value of k is bounded by $O(n)$ but believed to be constant for almost all parts. The algorithm for the design of the more restricted gaps by Berretty et al [1] has a worst-case running time of $O(n^2 \log n)$ for all parts. The algorithm reported in this paper can also report a gap trap in $O(n\alpha(n)\log n)$ time, which is a considerable improvement over the algorithm by Berretty et al.

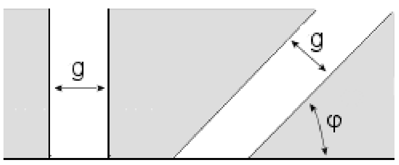

Fig. 4. A gap specified by its width g (left), and a guillotine trap specified by its width g and orientation φ (right).

We have implemented our complete algorithm for the design of guillotine traps using C++ and Mathematica, and only use analytical techniques to compute a solution. Various illustrations throughout the algorithmic section have been generated using this implementation.

We review our setting and problem statement. We consider rigid two-dimensional polygonal parts of a single type that move along a flat track in a quasi-static manner without interfering with each other. The radius of the helical track is assumed to be large compared to part dimensions, thus allowing us to approximate the section of the track as linear. As the part moves across a trap, gravity may cause it to fall into the hole. We also assume the position of the center of mass C is known, as it dictates the stability of the part. Furthermore, the feeder track is tilted to assure that the part rests in a *stable orientation* on the track's outer edge, i.e. the part rests on an edge of the convex hull that supports the center of mass. The input of our algorithm is a two-dimensional polygon P specified as a list of n vertices, coordinates of its center of mass C, and a preferred orientation α_{pref}. The output is a guillotine trap $T(\varphi, g)$ that rejects P in all orientations but α_{pref}, or a confirmation that no such trap exists.

This paper is organized as follows. In Section 2 we provide a geometric framework modeling various part characteristics; this model is the basis for our algorithms. This section furthermore sketches the outlines of our approach. In Section 3 we discuss the algorithm in detail and provide insight into the runtime analysis.

II. Trap Design

Let us first introduce some notation. We define the world coordinate frame such that the bottom of the track coincides with the horizontal x-axis. Throughout this paper, the direction of motion of P is in the positive x-direction (left to right). An additional coordinate frame is attached to P at its center of mass C. Similar as in [1], we express the *placement* of P with respect to the trap T by $P_\alpha(x)$; here x specifies the distance between C and the left trap edge, and $\alpha \in \mathcal{A}$ describes the orientation of the frame attached to P relative to the world frame. A shorthand we will frequently use is the notation P_α, which refers to P in a given orientation α.

A. Part safety

A part placement $P_\alpha(x)$ is *safe* if the part does not fall into the trap. To capture the notion of safety by a geometric criterion, we introduce the notion of a *support segment*. For a given angle $\beta \in \mathcal{B}$,‘ measured in the local frame of P, the support segment $S(\beta)$ is the portion of the line with orientation β through C between the two (boundary) points of P that are furthest apart. Note that $S(\beta) = S(\beta + \pi)$. We say that a support segment $S(\beta)$ is *active* for a given placement of P if its endpoints rest on the track on *opposite* sides of the trap

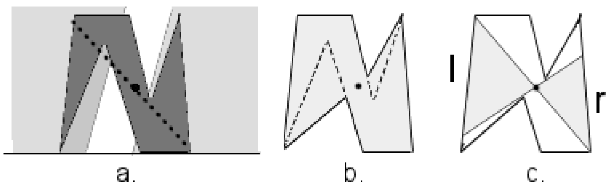

Fig. 5. (a) The dotted line is an active support segment for the dark polygon. (b) The star hull of for the polygon. (c) Support cone in with left face l and right face r.

(Fig. 5(a)). It follows from [1] that $P_\alpha(x)$ is safe if and only if there exists an active support segment for $P_\alpha(x)$.

We refer to the union $\bigcup_{\beta \in \mathcal{B}} S(\beta)$ of all support segments as the *star hull* of P (Fig. 5(b)) with center of mass C, as it is the smallest star-shaped [24] superset of P with C in its kernel. As described above, the safety of a placement of P is safe depends solely on the support segments of P. Consequently, since the star hull of P equals the union of its support segments, it is clear the following lemma holds. This lemma allows us to assume that P is star-shaped in the remainder of this paper.

***Lemma** 1:* A placement of P is safe if and only if the placement of the corresponding equivalent star hull is safe.

We define a *support cone* as the set of support segments with their endpoints on one specific pair of opposing edges (e_1, e_2) of P (Fig. 5(c)). A support cone consists of two halves, on opposite sides of C, one of which is bounded by a portion of e_1 and one of which is bounded by a portion of e_2. For two non-parallel edges, let p_i be the intersection of the two lines containing respectively e_1 and e_2. Now, taking the vector $\overrightarrow{Cp_i}$ as reference, we refer to the partial P-edge left of $\overrightarrow{Cp_i}$ as the cone's *left face*, and the one to right of $\overrightarrow{Cp_i}$ as its *right face*. A support cone, denoted as SC_i with $i \in \{1, \ldots, w\}$, corresponds to a continuous sub-interval of $\mathcal{B}$. We denote this sub-interval by $\mathcal{B}_i$. The following lemma follows immediately from the observation that every common boundary of two adjacent support cones ends at a vertex of the polygon P.

***Lemma** 2:* The number of support cones is linear to the number of vertices, i.e., $w = O(n)$.

We say that a support cone is *active* in a given placement, if it contains at least one active support segment.

Extending the definition of safety, a guillotine trap $T(\varphi, g)$ is safe for P_α, if for all $x \in [0, g]$ holds that placement $T(x, \alpha)$ is safe, or differently put, if there exists an active support segment $S(\beta)$ for all placements $P_\alpha(x), x \in [0, g]$. Throughout this paper, we will also use the alternative expression that P_α can safely pass $T(\varphi, g)$. Linking this to our goal, $T(\varphi, g)$ feeds $P_{\alpha_{\mathit{pref}}}$ if P can only safely cross $T(\varphi, g)$ in orientation α_{pref}.

B. General approach

An important notion in our approach is what we refer to as the *guillotine principle*: moving P_α over guillotine trap $T(\varphi, g)$ is in terms of safety exactly the same as moving $P_{\alpha+\Delta\phi}$ over $T(\varphi + \Delta\phi, g)$ for an arbitrary $\Delta\phi$; in fact, by adding $\Delta\phi$ to both the trap's and the part's orientation, we only change the frame of reference (Fig. 6). In practical terms, the guillotine principle implies that if P_α can safely cross guillotine trap T in orientation φ and width g, then $P_{\alpha+\Delta\phi}$ can safely cross guillotine trap T of the same width in orientation $\varphi + \Delta\phi$.

With the above in mind, the global approach to compute $T(\varphi, g)$ is as follows. The algorithm takes the guillotine trap in orientation $\varphi = \pi/2$ as a basis for its computations on. The algorithm starts off by computing *the safe function* $F^{\mathit{safe}}(\alpha)$, which specifies for each part orientation $T(\pi/2, g)$'s maximal trap width that the part can safely pass. Next, the algorithm identifies the part's orientation α_{max} at which F^{safe} has its maximum g_{max} (Fig. 7). Intuitively—if we temporarily disregard the fact that α_{max} is probably not a stable orientation—this means that $T(\pi/2, g_{max})$ feeds $P_{\alpha_{max}}$. Applying the guillotine principle, the algorithm's last step is to set the trap's orientation to $\pi/2 + \Delta\phi$, with $\Delta\phi = \alpha_{\mathit{pref}} - \alpha_{max}$, so that the preferred, stable orientation safely passes $T(\pi/2 + \Delta\phi, g_{max})$ (Fig. 7).

The first and major part of the next section deals with the generation of F^{safe}, which we will construct from a set of $O(n)$ trigonometric curves. The last part of the next section discusses the use of F^{safe} to construct the the desired guillotine trap. But before jumping to the next section, we first explain a few additional concepts. Note that because we take $T(\pi/2, g)$ as reference for the generation of $F^{\mathit{safe}}(\alpha)$, several definitions used for $F^{\mathit{safe}}(\alpha)$ will only be defined for $T(\pi/2, g)$.

C. Some additional notions

To construct $F^{\mathit{safe}}(\alpha)$, we need to distinguish three types of instabilities causing the part to fall: *forward-unsafety*, *backward-unsafety* and *topple-unsafety*.

- *Forward-unsafety*: The part falls forward into the trap, when C is positioned over the trap, but no part of P is supported by the portion of the track right of the trap.
- *Backward-unsafety*: The part falls backward into the trap, when C is positioned over the trap, but no part of P is supported by the portion of the track left of the trap.
- *Topple-unsafety*: Despite being supported by portions of the track left and right of the trap, the part P topples into the trap because there exists no active support segment.

Recall that a support segment $S(\beta)$ is active in a given part placement, if its endpoints rest on the track on opposite sides of the trap. In the case of $T(\pi/2, g)$, it is easy to see that only the horizontal distance between both endpoints of $S(\beta)$ is relevant to determine whether $S(\beta)$ is active. For a given P_α, we refer to this difference as the *projected length*, or *horizontal*

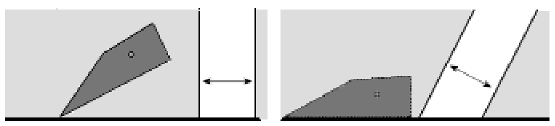

Fig. 6. Illustration of the guillotine principle.

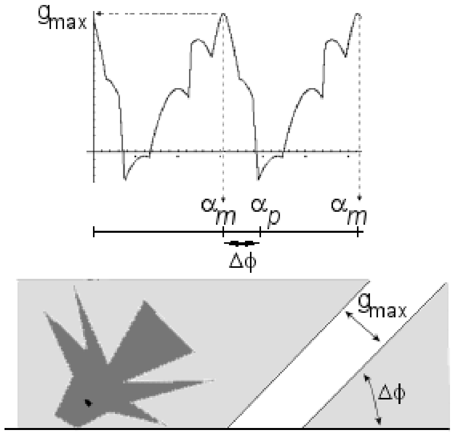

Fig. 7. The top figure depicts F^{safe} corresponding to the part in the bottom figure, which is shown in the preferred orientation (α_p). The interval just below the chart shows the relation between F^{safe}'s maxima α_m, α_p and $\Delta\phi$. Finally, the bottom figure shows the trap $T(\pi/2 + \Delta\phi, g_{max})$ that feeds the part in the preferred orientation.

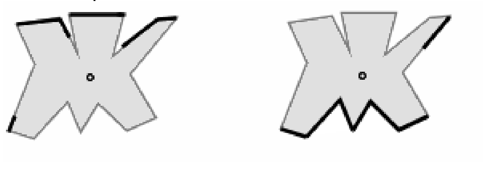

Fig. 8. Left and right figure respectively show the upper and lower envelope.

length, of $S(\beta)$, and denote it by $|S(\beta)|_\alpha$. Notice that $|S(\beta)|_\alpha$ is a function that takes both α and β as parameters.

Before we introduce the important notion of *critical* support segments, let us look at the concept of upper and lower envelope. The upper envelope of P is obtained by taking for each x-value the boundary point of P with the maximum y-value. In other words, a point p on the boundary of P belongs to the upper envelope if the vertical ray extending upward from P does not intersect P. The lower envelope is obtained in a similar fashion by taking the boundary points with minimum y-value [24] (Fig. 8). The following definition uses the envelopes to define a critical support segment.

***Definition** 1:* A support segment $S(\beta)$ is critical for P_α if both $S(\beta)$'s endpoints are either in the upper envelope or the lower envelope. For a part in given orientation α, we denote the set of critical support segments as $\mathcal{B}^{crit}(\alpha) \subset \mathcal{B}$.

This set of critical support segments $\mathcal{B}^{crit}(\alpha)$ features a special property: P_α passes $T(\pi/2, g)$ without toppling (although it may still fall forward or backward), if and only if each of its critical support segment becomes active for *at least* one placement $P_\alpha(x)$. This can be derived from [1]. Now, combining the above with the geometric criteria of activeness, we can conclude that P_α does not topple while passing $T(\pi/2, g)$, if and only if g is at most the minimum horizontal length of all $\mathcal{B}^{crit}(\alpha)$'s support segments.

Extending the criticality to support cones, we define a support cone to be critical for P_α when it contains at least one critical support segment. For a given P_α, we denote support cone $\mathcal{SC}_i$'s set of critical support segments by $\mathcal{B}_i^{crit}(\alpha) \subset \mathcal{B}_i$; i.e., for a given orientation α, $\mathcal{B}_i^{crit}(\alpha)$ specifies those support segments of $\mathcal{SC}_i$ that have both endpoints in either P_α's upper envelope or in its lower envelope. Hereby we observe that $\mathcal{B}_i^{crit}(\alpha)$ always specifies one continuous β-interval, and furthermore, that $\mathcal{B}_i^{crit}(\alpha) \neq \emptyset$ for two continuous sub-intervals of $\mathcal{A}$. These two sub-intervals correspond to P's orientations where both SC_i's faces are respectively (partially) in the lower envelope or upper envelope.

Combining the above, we come to the following definitions. For a given P_α, the set of critical support segments is the union of the critical support cones, thus $B^{crit}(\alpha) = \bigcup_{i=1}^{w} \mathcal{B}_i^{crit}(\alpha)$. Henceforth, P_α does not topple while passing $T(\pi/2, g)$, if and only if g is smaller or equal to the minimum of all critical support cones' minimum horizontal lengths.

***Lemma** 3:* A given part P_α can safely pass $T(\pi/2, g)$ if and only if $g \leq \min(M),\ where\ M = \bigcup_{i=1}^{k} \min\{|S(\beta)| \mid \beta \in B_i^{crit}(\alpha)\}$

III. Algorithm

For P_α to safely pass $T(\pi/2, g)$, it is necessary and sufficient that for all placements of P_α, the part does not fall forward or backward and does not topple. In this section, we define the *fall function* $F^{fall}(\alpha)$, specifying the maximum safe trap width that P_α can safely cross without falling forwards or backwards. Similarly, we define the *topple function* $F^{topple}(\alpha)$, specifying the maximum safe trap width that P_α can pass without toppling. The lower envelope of these two functions is the already introduced safe function $F^{safe}(\alpha)$.

The first and shortest subsection discusses the fall function, the second subsection then addresses the topple function, and the last subsection deals with the safe function and how the algorithm uses this curve to find the desired guillotine trap or report that no solution exists. In our discussion, we omit the proof of several complexity and time bounds for lack of space.

A. Fall function

Let us first look at the forwards unsafety. In a given orientation, the part falls into the trap when $T(\pi/2, g)$'s width is larger than the horizontal distance between C and the rightmost vertex. Hence, to prevent P_α from falling, the maximum safe trap width is equal to this horizontal distance. Similarly, the maximum safe trap width preventing the part to fall backwards is equal to the projected distance between C and the leftmost point. With this in mind, we can construct two functions using elementary geometry that map part orientation α to the respective maximum safe trap widths. The lower envelope of these two functions defines the function $F^{fall}(\alpha)$, specifying the maximum safe trap width preventing the part from falling both forwards and backwards.

***Lemma** 4:* F^{fall} is a continuous curve consisting of $O(n)$ simple differentiable pieces, and can be computed in $O(n)$ time.

B. *Topple function*

Recall that P_α does not topple while passing $T(\pi/2, g)$, if and only if g is at most the minimum horizontal length of all support segments in $\mathcal{B}^{crit}(\alpha)$. Consequently, this minimum horizontal length is the *maximum* topple-safe trap width, and should thus be the value $F^{topple}(\alpha)$ returns for a given P_α. Do recall here that this value can be calculated by taking the minimum of the P_α's critical support cones' minimum horizontal lengths (lemma 3). We define the minimum horizontal length of a support cone $\mathcal{SC}_i$ as the partially defined function $F_i^{topple}(\alpha)$, and thus $F^{topple}(\alpha) = \min\{F_1^{topple}(\alpha), \ldots, F_k^{topple}(\alpha)\}$. The remainder of this subsection will mainly discuss $F_i^{topple}(\alpha)$, and starts by introducing two building blocks for this discussion.

The *basis* of these blocks are F_i^{topple}'s α-intervals. A F_i^{topple} function is a partial function defined for two subintervals of $\mathcal{A}$, that is, while both $\mathcal{SC}_i$'s faces are (partially) in either the upper envelope or the lower envelope, or differently put, while $\mathcal{SC}_i$ is critical. Of these two, we will only discuss F_i^{topple} for the upper envelope, as the approach for the lower envelope is very similar. With this focus, only one relevant subinterval remains in which $\mathcal{SC}_i$ becomes critical. We refer to this interval as the *critical α-interval* $\mathcal{A}_i^{crit}$, allowing us to write $\mathcal{SC}_i$'s topple function as $F_i^{topple} : \mathcal{A}_i^{crit} \rightarrow \mathcal{B}_i^{crit}$.

In addition to $\mathcal{A}_i^{crit}$, we introduce the notion of *upward α-interval* $\mathcal{A}_i^{up}$. This upward α-interval is a continuous subinterval of $\mathcal{A}$ in which for $\mathcal{SC}_i$'s left as well as right face holds that their outward normal does *not* point downward. We say that orientation $\alpha \in \mathcal{A}_i^{up}$ is an *upward orientation*. Figure 9 illustrates $\mathcal{SC}_i$'s first and last upward orientation (a) and (b), as well as some intermediate upward orientation. An important observation is that (for concave parts) upward-ness is a necessary but not sufficient condition for critical-ness.

A brief notational remark before we continue is that while the already introduced F-functions are defined in the $(\mathcal{A}, \mathbb{R}^+)$-plane, the following G-functions will be defined in the $(\mathcal{A}, \mathcal{B})$-plane. This brings us to the *first* building block: G_i^{min}, which forms F_i^{topple}'s basis. We define $G_i^{min} : \mathcal{A}_i^{up} \rightarrow \mathcal{B}_i$ as the partial function that specifies for a given *upward* orientation $\mathcal{SC}_i$'s support segment with the smallest horizontal length, or more formally:

$$G_i^{min}(\alpha) = \arg\min\{|S(\beta)|_\alpha \mid \beta \in \mathcal{B}_i\})$$

Let us have a look at the shape of $G_i^{min}(\alpha)$. Orientations a and b in Fig. 9 show the first and last upward orientation of the support cone. In these figures, the fat support lines illustrate the support segments having smallest horizontal lengths. While rotating $\mathcal{SC}_i$ from this first to last orientation, the support segment with minimum horizontal length continuously shifts thought the support cone. This behavior is equal for all support cones and results in a G_i^{min} function which consists of one concave, one convex sub-curve, and two straight line segments. The G_i^{min} function can be constructed in constant time using basic trigonometry.

The *second* building block are the $\alpha\beta$-surfaces with which G_i^{min} interacts. Recall that $\mathcal{A}_i^{crit}$ specifies $\mathcal{SC}_i$'s critical orientations, and furthermore, that $\mathcal{B}_i^{crit}(\alpha)$ specifies for each of these critical orientations $\mathcal{SC}_i$'s set of critical support segments. Hence, $\mathcal{B}_i^{crit}(\alpha)$ defines a bounded surface in the $\alpha\beta$-plane which we refer to as $\mathcal{SC}_i$'s *critical surface*. The intuition behind the critical surface is that a point (α, β) in this surface tells us that support segment β is critical for P_α. In addition, observe that (α, β) can only be in the critical surface if $(\alpha, \beta) \in \mathcal{A}_i^{up} \times \mathcal{B}_i$; i.e., the critical surface is a sub-surface of $\mathcal{A}_i^{up} \times \mathcal{B}_i$. We refer to the latter as $\mathcal{SC}_i$'s *upward surface*. The relation between both surfaces is that we use the upward surface as the basis and, if necessary, constrain it to form the critical surface.

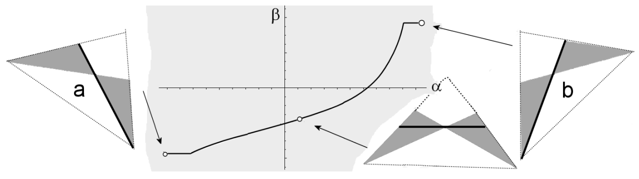

Fig. 9. This $\alpha\beta$-coordinate system shows G_i^{min} for the depicted triangle's support cone. For each of the three displayed cone orientations α, the fat line segment indicates the support segment of minimum horizontal length. The arrows identify the relation with G_i^{min}.

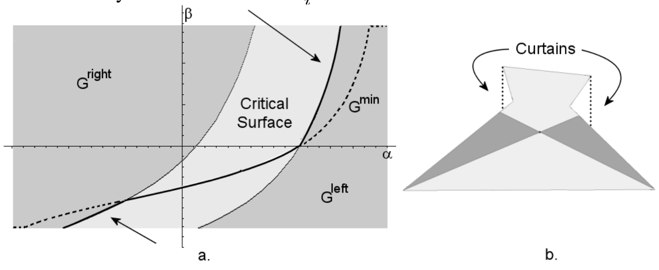

Fig. 10. **(a)**. The entire gray area (light and dark) in (a) is the upward surface corresponding to the support cone depicted in (b). G_i^{left} and G_i^{right} constrain this upward surface and so form the critical area (light gray). The function G_i^{min} is similarly formed as in figure 9, and is drawn as a dotted line while outside the critical area. For these α-intervals that G_i^{min} is not critical, the critical surface boundaries (in (a) identified with arrows) closest to G_i^{min} provide the closest-to-minimum β-values for F_i^{topple}. **(b)**. Shows the support cone in a critical orientation. In this orientation, the left face is unblocked and the right face is partially blocked.

Overall, the approach to construct F_i^{topple} is as follows: We first compute the two introduced building blocks, the critical surface and G_i^{min}. Next, we determine the interaction between both, and finally use this result to construct F_i^{topple}. Zooming in on this interaction, for the α-interval(s) in which G_i^{min} is *inside* the critical surface, the topple function $F_i^{topple}(\alpha)$ returns the horizontal length of support line $S(G_i^{min}(\alpha))$, while for α-values when G_i^{min} is *outside* the critical surface, $F_i^{topple}(\alpha)$ takes the critical β-value closest to $G_i^{min}(\alpha)$ to base its output on. An important observation is that this closest-to-G_i^{min} value is specified by the boundary of critical α-surface closest to $G_i^{min}(\alpha)$ (Fig. 10).

The construction of critical surface and its interaction with corresponding G_i^{min} is rather straightforward for convex parts, since for all $\mathcal{SC}_i$ applies that the critical surface is equal to the upward surface. This equality is a direct result of the fact that when SC_i is upward-orientated, both its faces are always in the upper envelope, and thus is SC_i then critical.

The critical surface
The difficulty for concave parts is that when a support cone is oriented upward, its faces can be (partially) "*blocked*" from the upper envelope by overhanging chunks of the part, preventing it from toppling. Note in this context that for a given orientation in $\mathcal{A}_i^{up}$, a support segment $\beta \in \mathcal{B}_i$ is critical if and only if its endpoints are not in the blocked sections of SC_i's faces. We model this phenomenon with two partial functions, the *left block function* $G_i^{left}(\alpha) : A_i^{up} \rightarrow \mathcal{B}_i' \subseteq \mathcal{B}_i$ and the *right block function* $G_i^{right}(\alpha) : A_i^{up} \rightarrow \mathcal{B}_i'' \subseteq \mathcal{B}_i$. The left block function specifies the unblocked subset of $\mathcal{B}_i$ for each $\alpha \in \mathcal{A}_i^{up}$, basing its output behavior only on SC_i's left face. Similarly, the right block function solely bases its output on what happens with the right face. Together, as illustrated in figure 10, the two functions constrain the upward surface, resulting in the critical surface.

For the moment, we skip the *construction* of G_i^{left} and G_i^{right} and focus on their *behavior* and *application*, starting with the *behavior* of G_i^{left}. Aside from the obvious case when the left face is unblocked for all its upward orientations, the behavior of G_i^{left} in terms of its $\mathcal{B}_i'$-interval is as follows: The cone's left face starts (partially) unblocked at SC_i's first upward orientation, which sets the initial content of $\mathcal{B}_i'$. Then while we increase the orientation α, the left face becomes (further) blocked, i.e. B_i' decreases. This process continues until the entire left face is blocked, i.e. $\mathcal{B}_i' = \emptyset$. The dark gray area in Fig. 10 indicated with G^{left} illustrates the growth of the blocked subset of $\mathcal{B}_i$, and thus indirectly the decrease of $\mathcal{B}_i'$.

The *behavior* of G_i^{right} is in a sense opposite to that of G_i^{left}: The right face starts entirely blocked, and thus $B_i'' = \emptyset$. While the orientation increases, $\mathcal{B}_i''$ increases as the unblocked region of the right face grows. The dark gray area in Fig. 10 indicated with G^{right} illustrates the decrease of the blocked subset of $\mathcal{B}_i$, and thus indirectly the growth of $\mathcal{B}_i''$.

As for *application* of the blocking functions, they "cut" the critical surface out of the upward surface by blocking areas of the upward surface (light gray area in figure 10). We are now able of formally establishing the relation between the upward and the critical surface: $\mathcal{B}_i^{crit} : \mathcal{A}_i^{crit} \rightarrow \mathcal{B}_i'$, where $\mathcal{A}_i^{crit} = \{\alpha \in \mathcal{A}_i^{up} \mid G_i^{left}(\alpha) \cap G_i^{right}(\alpha) \neq \emptyset\}$ and $\mathcal{B}_i' = \{\beta \in \mathcal{B}_i \mid \beta \in G_i^{left}(\alpha) \wedge \beta \in G_i^{right}(\alpha)\}$

We now zoom in on the *construction* of G_i^{left} (for G_i^{right} it is very similar). An important observation is that for SC_i's left face, instead of monitoring the entire set of overhanging edges and vertices, it suffices to keep track of the leftmost vertex hanging over the face. To construct $G_i^{left}(\alpha)$, we first need to identify the vertices that play a role as leftmost vertex for SC_i, and moreover, compute in which α-interval each of these vertex plays this role for SC_i. We will call such a tuple of leftmost vertex and α-interval a *leftmost unit*. The second step is to use the gathered leftmost units to construct of G_i^{left}.

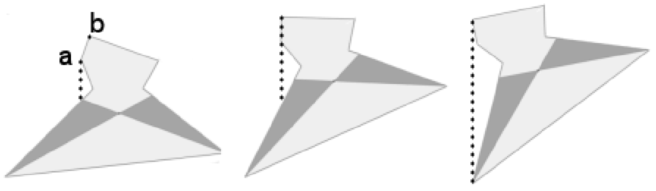

Fig. 11. In the first picture, vertex a starts as leftmost vertex and begins to sweep its curtain over the support cone's left face. Then in second picture a domination change occurs, that is, b dominates a. The last figure shows the completely blocked left face.

***Lemma** 5:* The number of leftmost units is $O(n)$.

To compute the leftmost units, we switch our perspective from support cones to vertices; that is, we compute per vertex for which support cones it is a leftmost vertex as well as the corresponding α-intervals. To accomplish this, we make three observations, while keeping in mind that we work with the star-hull of P: A vertex can only be leftmost if it is left of C (in the world coordinate system). Secondly, a vertex can only be a leftmost vertex if it is part of the upper envelope. And lastly, any vertex v_j is guaranteed to be in the upper envelope when straight above C, and to lose its place in the upper envelope again at some orientation α when some vertex v_i hangs straight above v_j. In the latter case, we say that v_i dominates v_j at α, and refer to the event as a *domination change* (Fig. 11).

With this in mind, the general idea of the leftmost-vertex algorithm is as follows: Rotate the part left around C. As soon as vertex v_i arrives straight above C, we "hang" a curtain l_i from v_i, i.e. a half-line pointing straight down, and add v_i to the status structure. While rotating further left, we keep track of which left faces l_i (partially) sweeps, and report v_i and the corresponding α-interval as a leftmost unit to the respective support cones. A domination change occurs each time a l_i hits a vertex v_j, hence v_i dominates v_j and we can eliminate v_j from the status structure. This algorithm runs until each vertex of P has been added and again removed from the status structure.

The second step in the construction of G_i^{left} is the conversion of the leftmost units into functions which together form G_i^{left}. Let $G_{i,j}^{left} : \mathcal{A}_{i,j}^{up} \rightarrow \mathcal{B}_i'$ with $\mathcal{A}_{i,j}^{up} \subset \mathcal{A}_i^{up}$ and $\mathcal{B}_i' \subseteq \mathcal{B}_i$ be such a function, where i and j respectively specify the support cone SC_i and the vertex v_j, and where $\mathcal{A}_{i,j}^{up}$ is the α-interval for which v_j is SC_i's leftmost vertex. The function $G_{i,j}^{left}$ can be constructed in constant time using elementary trigonometry, and is mainly based on v_j's vertical projection onto the left face. Finally, the construction of G_i^{left} is a straightforward combination of its leftmost unit functions.

***Lemma** 6:* The generation of all critical surfaces takes $O(n)$ time in total.

Interaction $G_i^{min}(\alpha)$ and critical surface
Now that we defined and constructed the critical surface for

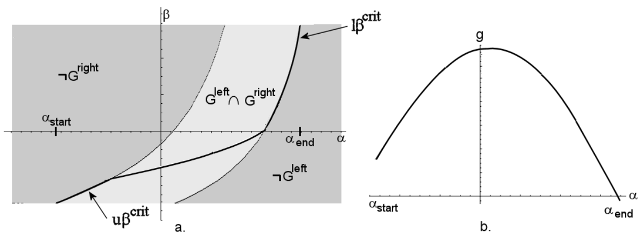

Fig. 12. The left figure shows the same scenario as figure 10. The right picture depicts the corresponding F_i^{topple}, where the g-value is the trap width.

concave parts, the next step is to combine it with the function $G_i^{min}(\alpha)$. For concave parts, we can clearly not guarantee that G_i^{min} is entirely in the critical surface. For a given $\alpha \in A_i$ we distinguish two scenarios: Firstly, $G_i^{min}(\alpha) \in \mathcal{B}_i^{crit}(\alpha)$, i.e. $\mathcal{SC}_i$'s support segment with the minimal horizontal length is critical, and thus $F_i^{topple}(\alpha) = |S(G_i^{min}(\alpha))|_\alpha$. In the second scenario $G_i^{min}(\alpha) \notin B_i^{crit}(\alpha)$, leaving us to select the critical β-value closest to the one specified by $G_i^{min}(\alpha)$. This β-value is specified by the critical surface boundary closest $G_i^{min}(\alpha)$. As illustrated by Fig. 12(a), the *lower-* and *upper β-boundary* of the critical surface are equal to respectively $\min G_i^{left}(\alpha)$ and $\max G_i^{right}(\alpha)$. Let us denote these bounds as $l\beta_i^{crit}(\alpha)$ and $u\beta_i^{crit}(\alpha)$.

Finally, before defining F_i^{topple}, let the complement of G_i^{left} be $\neg G_i^{left}(\alpha)$, which specifies the blocked subset of $\mathcal{B}_i$ for each $\alpha \in \mathcal{A}_i^{up}$, and similarly, let $\neg G_i^{right}(\alpha)$ be the complement of G_i^{right}. This allows us to define $F_i^{topple}(\alpha)$ as follows:

$$\begin{cases} |S(l\beta_i^{crit}(\alpha))|_\alpha, & G_i^{min}(\alpha) \in \neg G_i^{left}(\alpha) \\ |S(G_i^{min}(\alpha))|_\alpha, & G_i^{min}(\alpha) \in (G_i^{left}(\alpha) \cap \; G_i^{right}(\alpha)) \\ |S(u\beta_i^{crit}(\alpha))|_\alpha, & G_i^{min}(\alpha) \in \neg G_i^{right}(\alpha) \end{cases}$$

***Lemma** 7:* The cumulative complexity of all topple functions F_i^{topple}, $i = 1, \ldots, w$, is $O(n)$.

We conclude this subsection with a remark on the shape of F_i^{topple}. Fig. 12 shows a typical topple function i. All F_i^{topple} functions are similarly shaped, although they may just consist of an increasing or decreasing curve. Any pair of F_i^{topple} function can intersect at most twice.

Topple function $F^{topple}(\alpha)$
At his point, we can construct F^{topple} from the $O(n)$ "sub" topple functions. Recall that $F^{topple}(\alpha) = \min\{F_1^{topple}(\alpha), \ldots, F_k^{topple}(\alpha)\}$, i.e. F^{topple} is the lower envelope of the set of F_i^{topple} functions. Such an envelope can be calculated in $O(\lambda_3(n) \log n)$ time [25], where $\lambda_3(n) \leq O(n\alpha(n))$ and $\alpha(n)$ the functional inverse of the Ackermann's function.

***Lemma** 8:* F^{topple} is a function consisting of $O(n\alpha(n))$ simple differentiable pieces, and can be computed in $O(n\alpha(n) \log n)$ time.

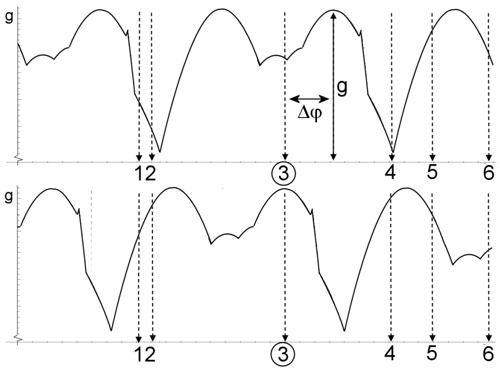

Fig. 13. The top chart shows the initial F^{safe}. The dotted arrows $1, \ldots, 6$ indicate the stable orientations, where 3 is the preferred one. From the bottom $\Delta\varphi$-shifted chart, we can conclude that $T(\pi/2 + \Delta\varphi, g)$ feeds P_3, for none of the remaining stable orientation other than 3 is associated with maximums.

C. Safe-function $F^{safe}(\alpha)$ and its application

As explained at the start of this section, the safe function $F^{safe}(\alpha)$ is the lower envelope of $F^{fall}(\alpha)$ and $F^{topple}(\alpha)$. These functions are respectively $O(n)$ and $O(\alpha(n)n)$ in complexity and their computation takes $O(n)$ and $O(n\alpha(n) \log n)$ time. Combining these bounds brings $F^{safe}(\alpha)$'s complexity and time bound to respectively $O(\alpha(n)n)$ and $O(n\alpha(n) \log n)$.

The next main algorithmic step is the usage of the $F^{safe}(\alpha)$ function to compute $T(\varphi, g)$ that feeds $P_{\alpha_{pref}}$. To avoid unclarity, note that with F^{safe}'s maximums, we only refer to equally highest tops. We give an intuitive explanation using Fig. 13. Remember that the idea is to set the guillotine trap's width and orientation such that only $P_{\alpha_{pref}}$ can safely pass the trap. This translates as follows to our figure: Firstly, F^{safe}'s maximum becomes the trap's width g_{max}, since this is the largest possible trap width P can safely cross. This leaves us to find the right orientation for the guillotine trap. To determine this, we first make the following observation, while keeping in mind that $F^{safe}(\alpha)$ is based on the guillotine trap in orientation $\varphi = \pi/2$: By changing T's orientation, $F^{safe}(\alpha)$ horizontally shifts over the interval $[0, 2\pi]$. This idea is based on the earlier explained guillotine principle. The next and last step now is as follows: choose φ, i.e. shift $F^{safe}(\alpha)$, such that one of F^{safe}'s maximums is associated with α_{pref}. In our figure, "associated" entails that this maximum and α_{pref} are on the same vertical line. On top of that, no other stable orientation may be associated with one of the remaining maximums, as otherwise $T(\varphi, g_{max})$ would be safe for multiple stable orientations. The guillotine trap is successfully constructed, when this demand is satisfied.

From the above it directly follows that $P_{\alpha_{pref}}$ can only be fed by a guillotine trap, if we can associate α_{pref} with one of F^{safe} maximums without simultaneously associating one of the other stable orientations with a remaining maximum. In the introduction we mentioned a class of parts for which the upper bound on the running time is $O(n^2)$. With the above in mind, it is clear that this is the case for parts with both $O(n)$

maximums and $O(n)$ stable orientations. This class contains highly symmetrical and regular parts.

Theorem 1: We can compute in $O(n\alpha(n)\log n + nk)$ time a guillotine trap that feeds $P_{\alpha_{pref}}$, where k is the number of maxima of $F^{safe}(\alpha)$, or report that no such trap exists.

IV. Conclusion

We presented a new class of traps for vibratory bowl feeders, receiving a set polygonal parts in an arbitrary stable orientation as input and outputting only parts in one preferred stable orientation. The approach has been implemented and tested using C++ and Mathematica. The advantage of the guillotine trap lies in the fact that it enables the selection of a preferred stable orientation, whereas previous classes of traps do not allow control over the orientation to be fed. As illustrated, this capability can be used to increase the feed rate. In addition, the proposed algorithm is considerably faster than the existing algorithm for the more constrained and less powerful gap, as it features a runtime of $O(n\alpha(n)\log n + nk)$. The $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann's function and k is the number of candidate solutions. The value of k is a constant for most parts, including polygonal parts without a high degree of symmetry and regularity.

The latter brings us to the extensions and open problems of our current work. One challenging open problem is to fully characterize the class of feedable parts, and to identify the parts that lead to a constant number k of candidate traps to be evaluated. We conjecture that only very regular parts give rise to non-constant k. In addition, to further improve the impact of our approach we intend to incorporate the vibration of the bowl into our model: in practice the part does not slide along the track, but makes little "jumps". As guillotine traps permit us to select the orientation to be fed, we feel that they provide a good starting point for the study of the effect of vibrations. It will be our goal to slightly adjust the trap so that it accepts all perturbed versions of the correctly oriented part, but still rejects all perturbed versions of wrongly oriented parts. One of the challenges will be to quantify the maximally allowed perturbations. The outcomes of this study will also be of value to the feeding of toleranced parts. A second practical issue lies in the fact that a part can get stuck in the trap when it topples. Hence one would like to be able avoid such an unproductive situation, or perhaps minimize the chance of such a thing happening.

Another interesting direction of research is the extension to three-dimensional parts. To our knowledge no research has been conducted in this area yet.

Acknowledgment

Onno C. Goemans acknowledges the support of the Netherlands Organisation for Scientific Research (NWO). A. Frank van der Stappen is in part supported by the Dutch BSIK/BRICKS-project.

References

[1] R.-P. Berretty, K. Goldberg, M. Overmars, and A. van der Stappen, "Trap design for vibratory bowl feeders," *International Journal of Robotics Research*, vol. 20, pp. 891–908, 2001.

[2] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski, "A complete algorithm for designing passive fences to orient parts," *Assembly Automation*, vol. 17, no. 2, pp. 129–136, 1997.

[3] K. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, 1993.

[4] S. Akella and M. Mason, "Posing polygonal objects in the plane by pushing," *IEEE ICRA*, pp. 2255–2262, 1992.

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

[6] M. Peshkin and A. Sanderson, "The motion of a pushed sliding workpiece," *IEEE Journal of Robotics and Automation*, vol. 4, no. 6, pp. 569–598, 1988.

[7] R.-P. Berretty, K. Goldberg, M. Overmars, and A. van der Stappen, "Computing fence designs for orienting parts," *Computational Geometry: Theory and Applications*, vol. 10, no. 4, pp. 249–262, 1998.

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

[9] S. Akella, W. Huang, K. Lynch, and M. Mason, "Parts feeding on a conveyor with a one joint robot," *Algorithmica*, vol. 26, no. 3, pp. 313–344, 2000.

[10] M. Erdmann and M. Mason, "An exploration of sensorless manipulation," *IEEE Journal of Robotics and Automation*, pp. 367–279, 1988.

[11] B. Natarajan, "Some paradigms for the automated design of parts feeders," *International Journal of Robotics Research*, vol. 8, no. 6, pp. 89–109, 1989.

[12] K.-F. Bohringer, V. Bhatt, B. Donald, and K. Goldberg, "Algorithms for sensorless manipulation using a vibrating surface," *Algorithmica*, vol. 26, pp. 389–429, 2000.

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

[14] 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*, vol. 32, no. 11, pp. 2513–2524, 1994.

[15] P. conversation with Warren Wilson of TechnaVibes, Placerville, CA, 2002.

[16] D. Berkowitz and J. Canny, "Designing part feeders using dynamic simulation," *IEEE ICRA*, 1996.

[17] M. Jakiela and J. Krishnasamy, "Computer simulation of vibratory part feeding and assembly," 2^{nd} *International Conference on Discrete Element Methods*, 1993.

[18] G. Maul and M. Thomas, "A systems model and simulation of the vibratory bowl feeder," 2^{nd} *International Conference on Discrete Element Methods*, vol. 16, no. 5, pp. 309–314, 1997.

[19] A. Christiansen, A. Edwards, and C. Coello, "Automated design of parts feeders using a genetic algorithm," *IEEE ICRA*, 1996.

[20] M. Caine, "The design of shape interactions using motion constraints," *IEEE ICRA*, 1994.

[21] P. Agarwal, A. Collins, and J. Harer, "Minimum trap design," *IEEE Conference on Robotics and Automation*, pp. 2243–2248, 2001.

[22] R.-P. Berretty, K. Goldberg, M. Overmars, and A. van der Stappen, "Geometric trap design for automatic part feeders," *International Symposium on Robotics Research*, 1999.

[23] A. van der Stappen, R.-P. Berretty, K. Goldberg, and M. Overmars, "Geometry and part feeding," in *Sensor Based Intelligent Robot Systems*, G. Hager, H. Christensen, H. Bunke, and R. K. Eds, Eds. Berlin: Springer Verlag, 2002, pp. 259–281.

[24] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry – Algorithms and Applications.* Springer Verlag, 1997.

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

[26] Y.-B. Chen and D. Ierardi, "The complexity of oblivious plans for orienting and distinguishing polygonal parts," *Algoritmica*, vol. 14, pp. 14–367, 1995.