On the Design of Traps for Feeding 3D Parts on Vibratory Tracks

Onno C. Goemans and A. Frank van der Stappen
Institute of Information and Computing Sciences, Utrecht University,
PO Box 80089, 3508 TB Utrecht, The Netherlands

Abstract

In the context of automated feeding (orienting) of industrial parts, we study the algorithmic design of traps in the bowl feeder track that filter out all but one orientation of a given polyhedral part. We propose a new class of traps that removes a V-shaped portion of the track. The proposed work advances the state-of-the-art in algorithmic trap design by extending earlier work [1, 6, 17]—which focuses solely on 2D parts—to 3D parts, and by incorporating a more realistic part motion model in the design algorithm. We exploit the geometric structure of the design problem and build on concepts and techniques from computational geometry to obtain an efficient algorithm that reports the complete set of valid traps.

1 Introduction

A *part feeder* takes in a stream of identical parts in arbitrary orientations and outputs them in a single orientation. We consider the problem of *sensorless orientation* of parts in which parts are positioned and oriented using passive mechanical compliance. The input is a description of the part and the output is a sequence of physical actions by preferably simple devices that move a part from its unknown initial orientation into a unique final orientation. Among the sensorless part feeders considered in literature are the parallel-jaw gripper [18, 32], the single pushing jaw [3, 24, 28], the conveyor belt with a sequence of (stationary) fences [5, 11, 32] or pins [34] placed along its sides, the conveyor belt with a single rotational fence [2], the tilting tray [15, 27], vibratory plates [29] and programmable vector fields [8].

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

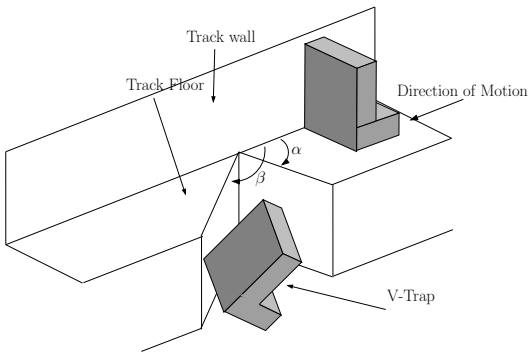

Figure 1: Vibratory track with V-trap $T(\alpha, \beta)$.

with (identical) parts, and a helical metal track that starts at the bottom and winds its way up along the bowl [9]. The bowl and track undergo an asymmetric helical vibration that causes the 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. A stream of oriented parts emerges at the top after successfully running the gauntlet. A device or sequence of devices is said to *feed* a part if it allows only one specific orientation of the part to pass.

For the manufacturing of feeders, engineers still rely mostly on skill, experience, and ad-hoc guidelines. Currently, the largest cost factor of bowl feeder production is the design of the custom mechanisms tailored to feed a specific part. The expenses and time associated with the design of part feeders remains a barrier to flexible automation. In a typical scenario, the feeder bowl takes up to 200 hours to manufacture; 95 % of this time is spent designing the bowl [17]. It is evident that automation of the design process would greatly reduce the production cost and time.

To aid in the design of bowl feeder layouts, researchers have used simulation [4, 21, 26, 30, 31], heuristics [22] and genetic algorithms [13]. Geometric analysis tools have been developed that help designers visualize the configuration space of a given combination of part and bowl layout [12]. Research to single reorientation and rejection mechanisms has been focused at certain classes of simple parts [9, 10] and, for a more general class of parts, at the design of traps [1, 6, 17]—traps are devices constructed by removing sections of the track. A common characteristic of the general solutions in the latter category is that the resulting (trap) designs only apply to two-dimensional parts. A recent piece of work not sharing this characteristic proposes a blade

mechanism for feeding polyhedral parts [16].

Even in the broader context of sensorless feeding in general, the majority of the achievements apply to flat, two-dimensional parts, or to 3D parts of which the resting face is fixed beforehand [25]. In addition to the aforementioned blade mechanism, three other contributions that also focus on polyhedral parts are work by Berretty *et al.* [7], which focuses on tilted plates with fences, work by Zhang and Gupta [33], which uses step devices, and work by Lynch [23], which employs a robot arm over a conveyor belt.

In this paper, we narrow the gap between the industrial application of vibratory bowls and scientific work on the automated design of sensorless feeders by studying the algorithmic design of traps for feeding 3D parts. Extending prior work on traps [1, 6, 17] for 2D parts, we propose a class of traps for feeding 3D parts that is inspired by similar devices from the practice of vibratory feeder design [9, 10]. We believe that this class of what we shall refer to as *V-traps* is rich enough to feed a broad class of 3D parts yet simple and structured enough to facilitate efficient automated design.

V-traps are created by removing a V-shaped section of the feeder track between two lines fanning out from the track wall. We characterize a V-trap by two parameters, α and β, which are the angles between the left and right trap edges, respectively, and the track wall (Fig. 1). A trap can thus be described as $T(\alpha, \beta)$. V-shaped traps provide a way to positively influence the deflection of rejected parts off the track: traps with a larger difference between α and β are preferable [9, 10]. The availability of all solutions, as output by our complete design algorithm, allows for the selection of trap designs with good deflection abilities as a post-processing phase. Except for the one-parameter balcony trap [6], we surmise that the class of V-traps generally provides stronger deflection capabilities than previously proposed classes of traps [1, 6, 17]; further research is needed to confirm this intuition.

In addition to presenting the first design algorithm for traps for feeding 3D parts, we advance the state of the art by incorporating a more realistic model for part motion into the design process. Prior work on the design of rejection devices for vibratory bowls assumes that parts slide smoothly along the track, while in reality they "hop" along the track. In this paper, we adopt a model for the hopping motion of parts as proposed by Boothroyd [9, 10] in the context of trap design, and extend our design algorithm to take this motion model into account. This extension allows us to eliminate trap designs that are undesirable for practical use.

The presented design algorithm for V-traps relies on various concepts and techniques from the field of computational geometry. The use of these concepts and techniques leads to a complete algorithm with provable effi-

ciency bounds. In a first phase, we apply a sweep-like approach [14, 19] to model the effect of the trap shape on the part-trap interaction. This phase results in a geometric arrangement of curves [14, 19] in the *trap space*, which is the two-dimensional space of all possible V-traps. In a second phase, we apply a divide-and-conquer approach to extract all valid trap designs. This set of valid solutions is defined by the so-called ≤ 1-level [14, 19] of the aforementioned geometric arrangement. Lastly, building on the results of the former two phases, we apply a sweep-like approach to identify the set of solutions that are also valid when considering a hopping part motion.

We consider rigid three-dimensional polyhedral parts of a single type, all of which are assumed identical. We assume the position of the center of mass C to be known, as it dictates the stability of the part. While moving up the track, a part rests on the track in a stable pose; that is, the part rests on the track floor and against the track wall, where both contacts support C. The stable floor contact is the result of gravity, while a slight tilt of the feeder track assures a stable wall contact. Finally, the part is rejected when its center of mass is no longer supported by the track floor.

Furthermore, as in the earlier works on algorithmic trap design, we assume these parts move along a flat track in a quasi-static manner without interfering with each other; also, we adopt the assumption that rejected parts are always deflected off the vibratory track. Although these assumptions still leave a gap to be bridged for the industrial application of automated trap design, we believe that the presented work provides a model flexible enough to allow for relaxing the aforementioned idealization. Finally, the radius of the helical track is assumed large compared to the dimensions of the part, thus allowing us to approximate the section of the track as linear.

Our goal in this paper is to develop an algorithm for the automated design of V-traps. The algorithm is complete in the sense that it identifies all existing valid trap designs that feed the part. It receives as input the polyhedral part P along with its center of mass C. The algorithm either outputs the set Σ of all valid trap designs that feed P or reports that no such trap exist. In an additional step, we also take the effective jump distance of parts as input, and output all valid trap designs that feed P while taking the more realistic motion model into account.

This paper is organized as follows. In section 2, we discuss the part-trap interaction in more detail and provide a number of definitions. In section 3, we first provide an intuitive explanation of our algorithm and continue with a discussion at a more detailed level. Section 4 explains the hopping part motion and discusses the necessary augmentations to the algorithm presented in section 3. We conclude in section 5 with a brief summary,

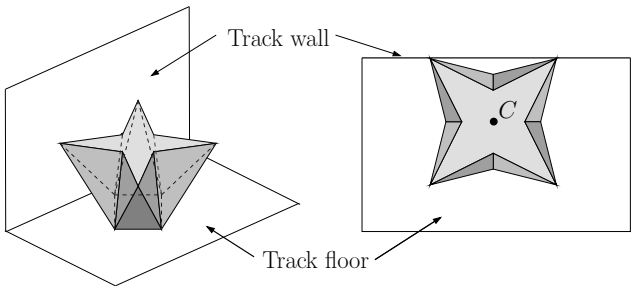

Figure 2: Side and top view of a stable track pose. Four possible wall contacts exist for the depicted floor contact; they correspond to the four edges of the rectangular convex hull of the projection of the pose onto the track floor.

several remarks and future work.

2 Modeling

We will first address the *track poses*, which are the possible poses of P whilst it moves up the track. The second subsection provides several general observations and definitions, and the third and final subsection discusses the interaction between part and trap.

2.1 Track Poses

A polyhedral part in three-dimensional space has three rotational degrees of freedom. While moving up the track, P settles in a stable pose on the track, thus discretizing its set of possible orientations. A stable track placement consists of two stable contacts, namely the contact with the (track) floor and with the (track) wall. We assume the floor and wall are sufficiently wide and high, respectively, such that the both always contain the perpendicular projection of P.

A stable floor contact is a placement in which P rests on the floor with a face of its convex hull that supports C. The *convex hull* [14] of P is the smallest convex set of points containing P or, informally, it is the polyhedron shaped by wrapping a rubber sheet around P. A face is said to support C if, when resting on the floor, the face contains C projected along the direction of gravity onto the floor. Observe that such a contact discretizes two of the

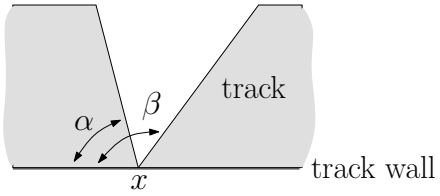

Figure 3: Illustration of a trap placement $T(\alpha, \beta, x)$.

three degrees of rotational freedom. The number of stable floor contacts is $O(n)$, where n is the number of vertices of P.

The second aspect of a stable pose, the stable wall contact, is ensured by the earlier mentioned track tilt. This contact discretizes the remaining degree of freedom, the roll orientation, which is the rotation about the axis through C perpendicular to the floor contact of P. Given P in a stable floor contact, the discretization of the roll orientation depends on the shape of the perpendicular projection of P and C onto the track floor; that is, a stable wall contact is a roll orientation in which the projection of P rests against the wall with an edge of its convex hull that supports the projected C (Fig. 2). For a given stable floor contact, the number of stable wall contacts is $O(n)$.

In the remainder of this paper, we refer to a stable part pose simply as a pose. As for each of the $O(n)$ floor contacts there can exist $O(n)$ wall contacts, the number of possible part poses is $O(n^2)$.

2.2 Definition & Notation

Let us first look in more detail at T. In theory, the domain of $\alpha \in A$ and $\beta \in B$ (Fig. 3) is $[0, \pi]$, where $\alpha < \beta$. In a practical setting, values in the lower and upper portions of these intervals produce traps that span too much track length to be practically feasible. We assume that the allowed α-interval, $[\alpha_s, \alpha_e]$, and β-interval, $[\beta_s, \beta_e]$, are given.

Let us consider P as it moves over T. The stability of a given pose positioned above T is determined by its floor features, which are the features of P resting on the floor when in the given pose. A pose is stable as long as the convex hull of the floor features contains the perpendicular projection of C onto the track floor. As the pose moves over T, each floor feature at some point temporarily loses contact with the floor. As a result, the pose can become unstable, in which case the pose is rejected by T.

In summary, for a given pose, the stability above T and thus rejection

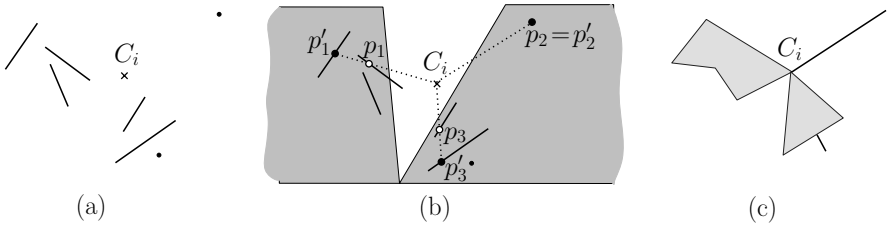

Figure 4: (a) set of floor features of P_i touching the floor. (b) illustrates that it suffices to consider p'_1, p'_2 and p'_3 with respect to the safety of the trap placement. (c) star-hull of P_i.

by T is determined by its floor features; hence, with respect to part-trap interaction, the design algorithm only needs to consider the floor features of a pose. This observation reduces our problem to a planar one with respect to part-trap interaction (Fig. 4a). We denote the floor features of a pose by P_i, where $i \in \{1, \ldots, k\}$ and, as shown before, $k = O(n^2)$. Furthermore, we denote by n_i the number of vertices of P_i. We note that P_i can consist of faces, edges or vertices of P, or any combination of the aforementioned; also, we remark that P_i can be free of the track wall. For a given P_i, let C_i denote C projected along the direction of gravity onto the track floor. We make the following remark on the complexity over all P_i.

Lemma 1. $\sum_{i=1}^{k} n_i = O(n^2)$.

Proof. An arbitrary edge e_a can occur as a floor feature for at most two floor contacts; each floor contact has $O(n)$ wall contacts, hence the total number of occurrences of e_a is bounded by $O(n)$. Summing over all $O(n)$ edges, the total complexity is $O(n^2)$. □

We define the world coordinate frame such that the wall coincides with the horizontal x-axis, and the positive y-axis crosses the floor. We note that $(x, 0)$ describes a point on the track wall. In practice, P_i slides across a stationary trap in the positive x-direction. It is, however, easier to describe our approach by viewing the part as stationary and sliding the trap underneath the part in the negative x-direction, which naturally is the same. We extend our trap notation and denote a *trap placement* by $T(\alpha, \beta, x)$, where $x \in X \subset \mathbb{R}$ specifies the position of the intersection point of both trap edges on the x-axis (Fig. 3). Furthermore, we denote a *left* and *right trap edge* by respectively $l(\alpha)$ and $r(\beta)$ and, similarly, use $l(\alpha, x)$ and $r(\beta, x)$ to denote the left and right trap edge for a particular trap placement.

2.3 Part-Trap Interaction

For a given $T(\alpha, \beta, x)$, we define the *supported subset* $S_i(\alpha, \beta, x)$ of P_i as the subset of P_i that is in contact with the track. Similarly, we define $S_i(\alpha, x)$ and $S_i(\beta, x)$ as the subsets of P_i that are in contact with the track adjacent to $l(\alpha, x)$ and $r(\beta, x)$, respectively. If the convex hull of $S_i(\alpha, \beta, x)$ contains C_i, then P_i is stable for $T(\alpha, \beta, x)$. We refer to such a placement as a *safe placement*; a placement is *unsafe* otherwise. For a given P_i, α and x, placement $T(\alpha, \beta_c, x)$ is a *critical placement* when $T(\alpha, \beta, x)$ is safe if and only if $\beta \leq \beta_c$. Alternatively, we say that β_c is the *critical* β for the given α and x.

Definition 1. *For a given P_i, α and x, the* critical β_c *is the β-value for which $T(\alpha, \beta, x)$ is a safe placement for $\beta \leq \beta_c$ and an unsafe placement for $\beta > \beta_c$.*

For a given P_i, a *safe trap* (design) $T(\alpha, \beta)$ is a trap for which each placement is safe; i.e., $T(\alpha, \beta)$ passes P_i without rejecting P_i. For a given P_i and α, we say that $T(\alpha, \beta_c)$ is a *critical trap* if it is a safe trap that features at least one critical placement; hence, $T(\alpha, \beta)$ is a safe trap for P_i for $\beta \leq \beta_c$, while unsafe for $\beta > \beta_c$. Again, we alternatively say that β_c is the critical β for the given α.

Definition 2. *For a given P_i and α, the* critical β_c *is the β-value for which $T(\alpha, \beta)$ is a safe trap for $\beta \leq \beta_c$ and an unsafe trap for $\beta > \beta_c$.*

The *star-hull* of P_i captures the part of P_i that is relevant to the safety of P_i over an arbitrary trap. The star-hull is defined by $\bigcup_{p \in P_i} \overline{C_i p}$ or, more informally defined, it is the set of points of P_i furthest away from C_i in every direction. Fig. 4a and fig. 4c show an example P_i and its star-hull, respectively.

The relevance of the star-hull can be explained as follows. Let us view a safe trap placement in a slightly different way. If we consider an arbitrary $T(\alpha, \beta, x)$ and P_i, then $T(\alpha, \beta, x)$ is a safe placement if and only if there exist three points p_1, p_2, p_3 in $S_i(\alpha, \beta, x)$, such that C_i is contained in $\Delta(p_1, p_2, p_3)$; the latter denotes the triangle with corners p_1, p_2, p_3 (Fig. 4b). Let us now consider extending a half-line from C_i through each of these three points, followed by the expansion of $\Delta(p_1, p_2, p_3)$ to the outermost intersections p'_1, p'_2 and p'_3 of each ray with P_i. Clearly, p'_1, p'_2 and p'_3 are also in $S_i(\alpha, \beta, x)$ and C_i is in $\Delta(p'_1, p'_2, p'_3)$; hence, $T(\alpha, \beta, x)$ is also safe for p'_1, p'_2 and p'_3. Moreover, this will be the case for any x for which $\Delta(p_1, p_2, p_3)$ supports $P_i(x)$. Thus, as we are only concerned with part safety, we can discard

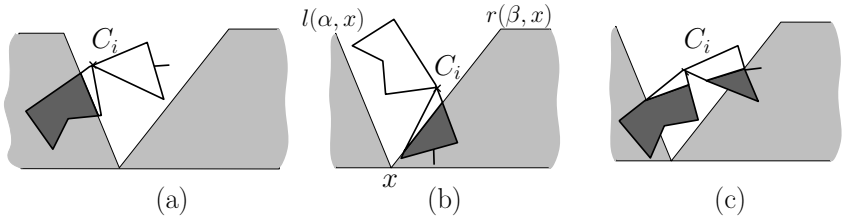

Figure 5: P_i is (a) forward-unsafe, (b) backward-unsafe and (c) topple-unsafe.

p_1, p_2, p_3 in favor of p'_1, p'_2, p'_3. We generalize this concept by extending a half-line from C_i in every direction and maintaining only the outermost intersection points with P_i. We assume for the remainder of this paper that each P_i is represented by its star-hull. We note that the star-hull has the same asymptotic complexity as P_i.

Definition 3. *The star-hull of P_i is defined as $\bigcup_{p \in P_i} \overline{C_i p}$.*

We can distinguish three types of unsafety that can cause P_i to fall into T: *forward-unsafety*, *backward-unsafety* and *topple-unsafety*.

- *Forward-unsafety*: P_i falls forward (Fig. 5a) when C_i is positioned over T, but P_i is not yet supported by $r(\beta, x)$.
- *Backward-unsafety*: P_i falls backward (Fig. 5b) when C_i is positioned over T, but P_i not is supported by $l(\alpha, x)$.
- *Topple-unsafety*: P_i topples (Fig. 5c) when C_i is positioned over T and P_i is in contact with both $l(\alpha, x)$ and $r(\beta, x)$, but C_i is unsupported.

3 Trap Design

In the first subsection, we provide an intuitive sketch of our algorithm. The second and third subsection model falling and toppling. The last subsection combines these two models to identify the set of all valid solutions Σ.

3.1 General Approach

A specific V-trap is defined by α and β. These parameters span a two-dimensional space, which we refer to as the *trap space*; each point in this space describes a unique V-trap. The idea is to subdivide the trap space for

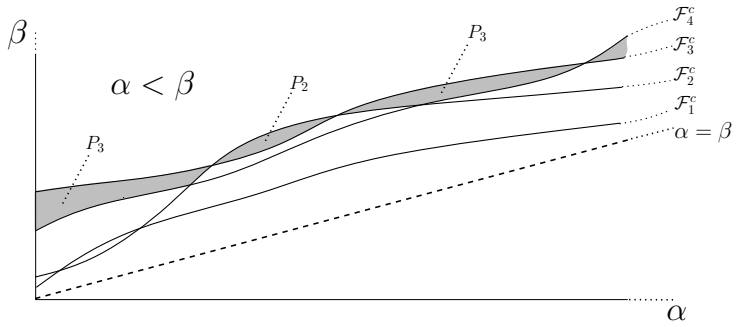

Figure 6: This figure shows four arbitrary critical trap functions in a subset of the trap space (these example functions do not match any real part). The light gray area depicts Σ, where the fed poses are specified for three subsets of Σ.

each P_i into a safe and unsafe region, where a (un)safe region is a set of points corresponding to (un)safe traps for P_i. We then combine the subdivisions of all P_i and extract Σ.

The subdivision for a given P_i is defined by the *critical trap function* $\mathcal{F}_i^c(\alpha)$, which specifies β_c for each α in $[\alpha_s, \alpha_e]$. The curve specified by this function divides the trap space in two subsets: points below or on $\mathcal{F}_i^c$ correspond to safe traps for P_i, while points above $\mathcal{F}_i^c$ correspond to traps that reject P_i.

The subdivisions are combined as follows. We add $\mathcal{F}_i^c$ for all $i \in 1, \ldots, k$ to the trap space, resulting in an arrangement of curves [14, 19]. The trap designs that reject all but one stable pose correspond to the points in the trap space that are above all but one $\mathcal{F}_i^c$. Let us refer to such a point as a valid solution σ and to the set of all valid solutions as Σ (Fig. 6).

Let us summarize, for this concept plays a central role in our algorithm. Each $\mathcal{F}_i^c$ corresponds to one specific P_i and specifies the impact of all possible trap designs on P_i. There exist $O(n^2)$ poses, thus the trap space contains $O(n^2)$ critical trap curves. By combining all $\mathcal{F}_i^c$, we are able to extract Σ.

We zoom in onto the critical trap functions, which are the building blocks of the algorithm. The computation of $\mathcal{F}_i^c$ consists of several algorithmic steps. Recall that we distinguish between forward-, backward- and topple-unsafety. We capture forward and backward-unsafety with the *fall function* $\mathcal{F}_i^f(\alpha)$, and we capture the topple-unsafety with the *topple function* $\mathcal{F}_i^t(\alpha)$. The function $\mathcal{F}_i^c$ is a composition of $\mathcal{F}_i^t$ and $\mathcal{F}_i^f$. These, as well as other functions defined in the following, are piecewise algebraic of bounded degree.

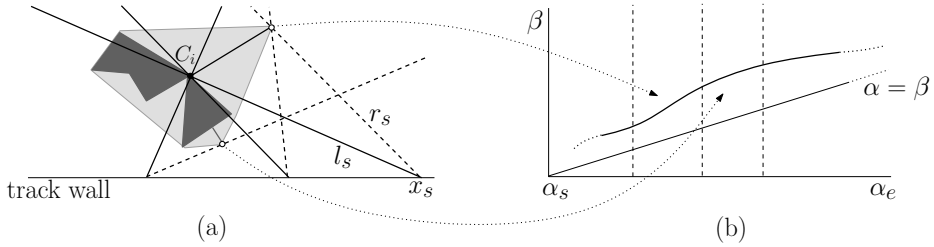

Figure 7: (a) l (solid lines) for three x values and r (dotted lines) for the corresponding β_f. Note that l_s and r_s are shorthands for $l(\alpha_s, x_s)$ and $r(\beta_f, x_s)$, respectively. (b) illustrates the shape of a resulting graph—the graph does not match the part. The arrows between fig. (a) and (b) illustrate the relation between two convex hull points and their corresponding fall curve segments.

Observation 1. *Given a P_i, critical trap function $\mathcal{F}_i^c$ subdivides the trap space in a set of safe and unsafe trap designs. $\mathcal{F}_i^f$ and $\mathcal{F}_i^t$ subdivide the trap space in subsets of traps that are respectively forward- and backward-(un)safe and topple-(un)safe.*

3.2 Falling

The fall function $\mathcal{F}_i^f$ specifies β_f for each α in $[\alpha_s, \alpha_e]$ such that P_i falls (does not fall) forward or backward into $T(\alpha, \beta)$ for $\beta > \beta_f$ ($\beta \leq \beta_f$); i.e., intuitively, all points below or on $\mathcal{F}_i^f$ correspond to trap designs that are forward- and backward-safe for P_i. The $\mathcal{F}_i^f$ function is composed of $\mathcal{F}_i^{ff}(\alpha)$, which captures forward falling, and $\mathcal{F}_i^{fb}(\alpha)$, which captures backward falling. In the following, we discuss the approach to construct $\mathcal{F}_i^{ff}$—we omit the similar construction of $\mathcal{F}_i^{fb}$—and conclude with the formation of $\mathcal{F}_i^f$.

Let us first consider α_s and identify the corresponding β_f. We choose $x = x_s$ such the left trap edge $l(\alpha_s, x_s)$ contains C_i; at this point, C_i is about to start moving over T, hence the chance on forward falling is maximal. We then choose β_f such that the right trap edge $r(\beta_f, x_s)$ is tangent to the right side of P_i (Fig. 7a). Clearly, choosing a larger β results in P_i falling forward into $T(\alpha, \beta)$, while a smaller β would not be a boundary value between the forward-safe and unsafe β's.

The approach to construct $\mathcal{F}_i^{ff}$ is as follows. Starting at α_s, we apply a sweep approach [14] by increasing α until $\alpha = \alpha_e$, while keeping C_i contained in l and maintaining r as the tangent to P_i. As a result, r "slides" along the

convex hull of P_i. Using basic geometry, we map each convex hull vertex encountered by r to a curve of constant degree; each curve defines $\mathcal{F}_i^{ff}$ for a sub-interval of $[\alpha_s, \alpha_e]$.

Lastly, to construct $\mathcal{F}_i^{f}$, we combine $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$ by taking the lower envelope of $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$. The lower envelope is the geometric construction comprised of all (partial) curves that are minimal with respect to β [14, 19]; i.e., all points below or on both $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$ correspond to trap designs that are forward- and backward-safe.

Lemma 2. *The functions $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$ consist of $O(n_i)$ curves of constant degree and require $O(n_i)$ computation time.*

Proof. Let us consider $\mathcal{F}_i^{ff}$—$\mathcal{F}_i^{fb}$ has the same asymptotic complexity. The function $\mathcal{F}_i^{ff}$ consists of a set of partial functions, where each partial function corresponds to a convex hull vertex. With the sweep approach in mind, it is clear that r can be tangent to each convex hull vertex only for one continuous α-interval. As there are $O(n_i)$ convex hull vertices, $\mathcal{F}_i^{ff}$ thus consists of $O(n_i)$ partial functions. Furthermore, each partial function requires constant time to construct, hence the overall computation requires $O(n_i)$ time. □

Lemma 3. *The function $\mathcal{F}_i^{f}$ consists of $O(n_i)$ curves of constant degree and requires $O(n_i)$ computation time.*

Proof. We add $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$ together in the trap space and subdivide $[\alpha_s, \alpha_e]$, such that each α-interval of the trap space contains exactly one partial function of both $\mathcal{F}_i^{ff}$ and $\mathcal{F}_i^{fb}$. The result is a set of $O(n_i)$ α-intervals that each contain two partial functions of constant degree. A pair of constant degree algebraic functions can at most have a constant number of intersections; hence, per α-interval, the complexity of the lower envelope is constant. Consequently, the complexity over all intervals is $O(n_i)$. Furthermore, the time costs per α-interval are constant, hence the overall computation requires $O(n_i)$ time. □

3.3 Toppling

The topple function $\mathcal{F}_i^{t}$ specifies β_t for each α in $[\alpha_s, \alpha_e]$ such that P_i does not topple into $T(\alpha, \beta)$ if and only if $\beta \leq \beta_t$; intuitively, P_i does (not) topple into traps corresponding to points (below or on) above $\mathcal{F}_i^{t}$. We recall that topple-unsafety can per definition only occur when C_i is over $T(\alpha, \beta, x)$ and both $l(\alpha, x)$ and $r(\beta, x)$ intersect P_i. In the remainder of this subsection, we will implicitly assume that $T(\alpha, \beta, x)$ satisfies these criteria.

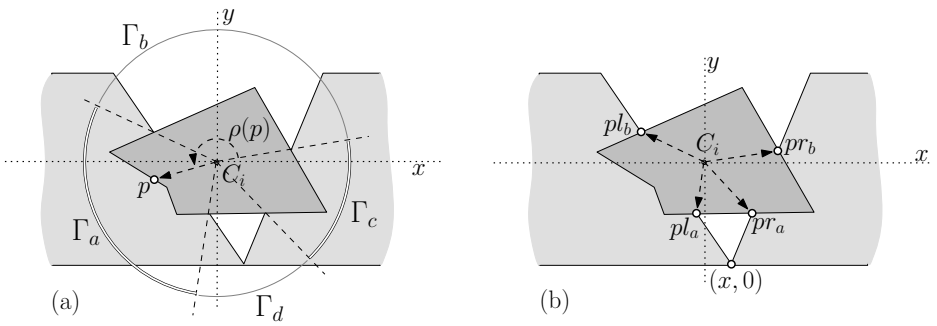

Figure 8: (a) Part plus local coordinate system with C_i as origin, where $\rho(p)$ specifies the angle in Γ for point p, and intervals Γ_a, Γ_c and Γ_b, Γ_d are supported and unsupported subsets of Γ, respectively. (b) The intersection points pl_a and pl_b of $l(\alpha, x)$ with P_i, and pr_a and pr_b of $r(\beta, x)$ with P_i.

Let us start with an alternative view on the definition of a safe part placement. We define a fixed local coordinate system with C_i as its origin. Let $\rho(p) \in \Gamma = [0, 2\pi]$ be the counter-clockwise angle of the half-line extending from C_i through point $p \in P_i$ with the positive x-axis of this local coordinate system (Fig. 8a). For a given $T(\alpha, \beta, x)$, we say that $\gamma \in \Gamma$ is *(un)supported* if there exists (no) $p \in S_i(\alpha, \beta, x)$ such that $\rho(p)$ equals γ. A placement $T(\alpha, \beta, x)$ is safe if and only if there exists no interval $[\gamma_a, \gamma_b]$ of length $|\gamma_a, \gamma_b| > \pi$ that is unsupported, where $|\gamma_a, \gamma_b|$ specifies the length of the γ-interval measured in counter-clockwise direction from γ_a. Furthermore, an interval is considered unsupported if it contains no supported γ (Fig. 8a).

With the above definition of a safe placement in mind, let us revisit the topple-unsafety definition. For an arbitrary $T(\alpha, \beta, x)$, let pl_a and pl_b be the intersection points of $l(\alpha, x)$ with P_i with the smallest and largest distance to $(x, 0)$, respectively; furthermore, let pr_a and pr_b be defined similarly for $r(\beta, x)$ (Fig. 8b). We will refer to pl_a and pr_a as the *first intersection point* of $l(\alpha, x)$ and $r(\beta, x)$ with P_i; similarly, we will refer to pl_b and pr_b as the *last intersection points*.

Using these points, we can define four closed γ-intervals of which the extremes are certain to be supported, namely $[\rho(pr_a), \rho(pr_b)]$, $[\rho(pr_b), \rho(pl_b)]$, $[\rho(pl_b), \rho(pl_a)]$ and $[\rho(pl_a), \rho(pr_a)]$. We observe that $|\rho(pr_a), \rho(pr_b)| < \pi$ and $|\rho(pl_b), \rho(pl_a)| < \pi$. Thus, to determine whether $T(\alpha, \beta, x)$ is safe, we only need to consider $[\rho(pr_b), \rho(pl_b)]$ and $[\rho(pl_a), \rho(pr_a)]$.

We note that the interiors of these two intervals contain no supported γ.

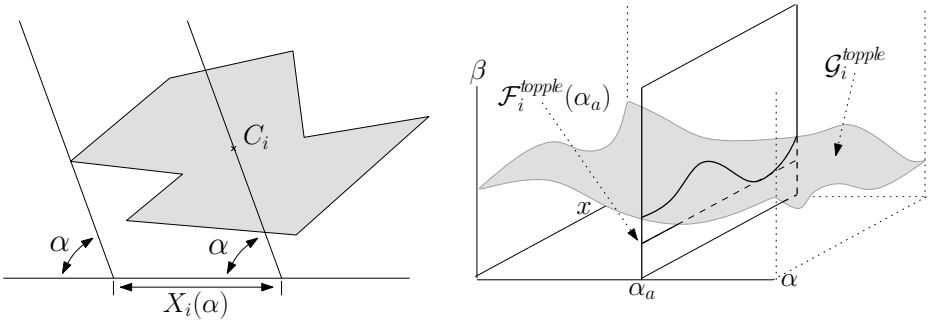

Figure 9: (a) an example of $X_i(\alpha)$, (b) illustration of mapping of $\mathcal{G}_i^{tb}$ to $\mathcal{F}_i^t$.

This note follows from the fact that P_i is represented by its star-hull (Def. 3), which can be seen as follows. Let us assume that γ_c in $[\rho(pr_b), \rho(pl_b)]$ is supported. Consequently, p_c with $\rho(p_c) = \gamma_c$ is either in $S_i(\alpha, x)$ or $S_i(\beta, x)$; without loss of generality, we can assume the latter. It then follows that $r(\beta, x)$ intersects $\overline{C_i p_c}$; let i_{new} denote their intersection. The distance from $(x, 0)$ to i_{new} must be larger than to pr_b. This, however, contradicts the definition of pr_b.

We can now formulate topple-unsafety as follows.

Lemma 4. *A trap placement $T(\alpha, \beta, x)$ is topple-unsafe if and only if $|\gamma(pl_a), \gamma(pr_a)| > \pi$ or $|\gamma(pr_b), \gamma(pl_b)| > \pi$. The points pl_a and pl_b are the first and last intersection point of $l(\alpha, x)$ with P_i; pr_a and pr_b are defined similarly for $r(\beta, x)$.*

Following the above definition, we can distinguish between two types of topple-unsafety, corresponding to the event of $|\gamma(pl_a), \gamma(pr_a)| > \pi$ and $|\gamma(pr_b), \gamma(pl_b)| > \pi$. Intuitively, in the latter case, P_i rotates away from the wall and flips directly into the bowl, while in the former case, P_i attempts to rotate toward the wall and thus slides down along the wall back into the bowl. We model both type of topple-unsafety separately by constructing the functions $\mathcal{F}_i^{ta}$ and $\mathcal{F}_i^{tb}$, after which both functions are combined to form $\mathcal{F}_i^c$. We focus our discussion on $\mathcal{F}_i^{tb}$, which captures topple-unsafety related to pl_b and pr_b; the explanation of $\mathcal{F}_i^{ta}$ is similar and will be limited to a few remarks.

In contrast to the approach used to construct $\mathcal{F}_i^f$, we need to consider a range of x-values to determine $\mathcal{F}_i^{tb}(\alpha)$ for a given α. We introduce the *placement space*, which is a three dimensional space spanned by α, β and x; observe that each point in this space specifies a unique trap placement.

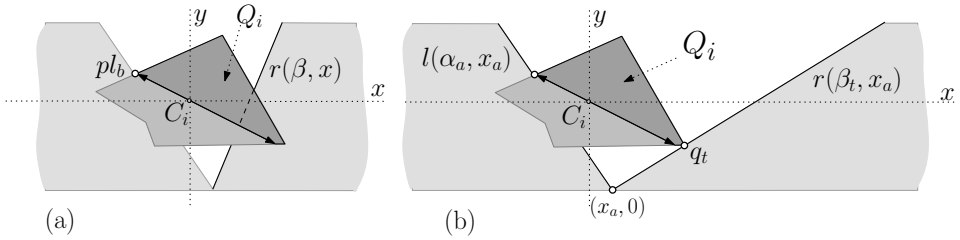

Figure 10: (a) Example of Q_i that is intersected by $r(\beta, x)$. (b) Example of identification of β_t for (α_a, x_a)—$r(\beta_t, x_a)$ is tangent to Q_i in point q_t.

In the placement space, we will construct an "extended" version of $\mathcal{F}_i^{tb}$ that takes x into account; we will denote this function by $\mathcal{G}_i^{tb}(\alpha, x)$. For a given α and x, this function specifies β_t such that P_i does not topple into $T(\alpha, \beta, x)$ if and only if $\beta \leq \beta_t$. Intuitively, $\mathcal{G}_i^{tb}$ describes a surface in the placement space such that P_i does (not) topple into traps corresponding to points (below or on) above $\mathcal{G}_i^{tb}$.

Let us first consider the domain of $\mathcal{G}_i^{tb}$. Recall that toppling can only occur when C_i is over T and P_i is in contact with both $l(\alpha, x)$ and $r(\beta, x)$. Applied to $\mathcal{G}_i^{tb}$, this means that the domain of $\mathcal{G}_i^{tb}$ consists of all (α, x) for which $l(\alpha, x)$ intersects P_i and C_i is on the trap-side of $l(\alpha, x)$. We use $X_i(\alpha)$ to denote the set of x-values of this domain for a given α (Fig. 9). Let us consider $\mathcal{G}_i^{tb}$ and $\mathcal{F}_i^t$ again. Clearly, P_i does not topple into $T(\alpha, \beta)$ when it does not topple into $T(\alpha, \beta, x)$ for $x \in X_i(\alpha)$. Hence, for a given α, $\mathcal{F}_i^t$ is defined as the minimum of $\mathcal{G}_i^{tb}$ for $x \in X_i(\alpha)$ (see Fig. 9).

Definition 4. $\mathcal{F}_i^t(\alpha) = \min\{\mathcal{G}_i^{tb}(\alpha, x) | x \in X_i(\alpha)\}$.

In summary, $\mathcal{G}_i^{tb}$ is the basis for the construction of $\mathcal{F}_i^{tb}$. In the following, we first discuss the identification of β_t for a given (α, x)-value, after which we continue by generalizing this concept and developing an algorithm to construct $\mathcal{G}_i^{tb}$. We conclude with the generation of $\mathcal{F}_i^{tb}$ using $\mathcal{G}_i^{tb}$.

We consider an arbitrary (α, x), denoted by (α_a, x_a), and determine the corresponding β_t. Let pl_b and pr_b be defined as before and let Q_i be defined as $\{p \in P_i | \rho(p) \in [\rho(pl_b) - \pi, \rho(pl_b)]\}$ (Fig. 10a). Based on lemma 4, we can conclude $T(\alpha_a, \beta, x_a)$ is topple-safe if and only if β is such that $r(\beta, x_a)$ intersects Q_i. Intuitively, Q_i contains all $p \in P_i$ for which $|\rho(p), \rho(pl_b)| \leq \pi$, hence β must be such that $r(\beta, x_a)$ contains a point of Q_i to ensure P_i does not topple into $T(\alpha_a, \beta, x_a)$. Upon reflection, we can conclude that β_t is such that $r(\beta_t, x_a)$ is tangent to Q_i. Let $q_t \in Q_i$ be the point where $r(\beta_t, x_a)$

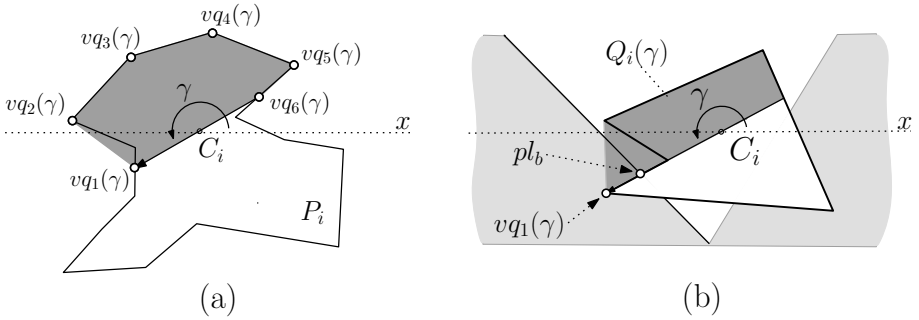

Figure 11: (a). $Q_i(\gamma)$ for a given γ, where $vq_2(\gamma), \ldots, vq_5(\gamma)$ are fixed vertices of P_i and $vq_1(\gamma)$ and $vq_6(\gamma)$ depend on γ. (b). Example of an event for which $vq_1(\gamma) \neq pl_b$: point pl_b is the last intersection point of $l(\alpha, x)$, but is not equal to $vq_1(\gamma)$.

touches Q_i (Fig. 10b). Clearly, for any $\beta > \beta_t$, trap edge $r(\beta, x_a)$ does not intersect Q_i, while for any $\beta < \beta_t$, trap edge $r(\beta, x_a)$ also intersecting $\overline{C_i q_t}$. As a final observation, we note that the convex hull of Q_i suffices to determine the tangent $r(\beta_t, x_a)$.

Definition 5. *$Q_i(\gamma) = \mathcal{CH}(\{p \in P_i | \rho(p) \in [\gamma - \pi, \gamma]\})$, where $\mathcal{CH}$ denotes the convex hull. Furthermore, let $vq_1(\gamma), vq_2(\gamma), \ldots, vq_m(\gamma)$ denote the vertices of $Q_i(\gamma)$, where m specifies the number of vertices in $Q_i(\gamma)$. The vertices are numbered in counter-clockwise order and $\rho(vq_1(\gamma)) = \gamma$.*

Let us further investigate $Q_i(\gamma)$ as this structure plays a central role in our approach. Firstly, we note that $vq_2, \ldots, vq_{m-1}$ are vertices of P_i and, except for their addition to and removal from $Q_i(\gamma)$, do not depend on γ. The position of vertices $vq_1(\gamma)$ and $vq_m(\gamma)$ does depend on γ (Fig. 11a). Secondly, when again considering (α_a, x_a), we can observe that pl_b is part of $Q_i(\gamma)$; more precise, their relation can be described as follows: $pl_b = vq_1(\rho(pl_b)) = vq_1(\gamma)$. For here on, we use $vq_1(\gamma)$ to refer to the last intersection point of $l(\alpha, x)$ with P_i. We note that there exists one exception to the latter note. That is, pl_b is not necessarily equal to $vq_1(\gamma)$ when the support line of an adjacent edge of $vq_1(\gamma)$ contains C_i (Fig. 11b). In the following discussion of the algorithm, we temporarily ignore such cases and discuss its treatment separately later on.

We generalize the use of $Q_i(\gamma)$. Let us consider Q_i for an arbitrary γ, denoted by $Q_i(\gamma_a)$. Employing $Q_i(\gamma_a)$, we can determine β_t for all (α, x) for which $vq_1(\gamma_a)$ is the last intersection point of $l(\alpha, x)$ with P_i. Intuitively,

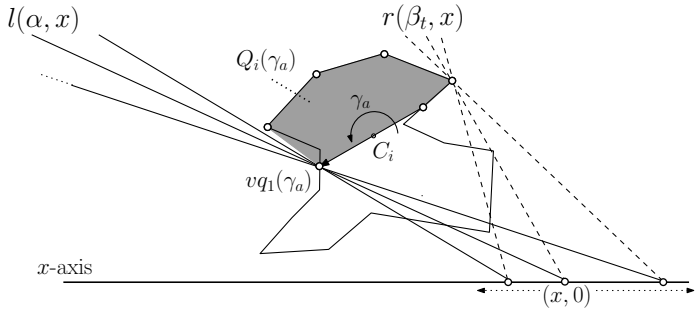

Figure 12: By sliding $(x, 0)$ along the x-axis and maintaining $l(\alpha, x)$ through $vq_1(\gamma_a)$, construct $Q_i(\gamma_a)$ can be used to identify β_t for a set of (α, x) values.

this set of (α, x) values can be discovered by sliding a point $(x, 0)$ along the x-axis, while maintaining $l(\alpha, x)$ through $vq_1(\gamma_a)$ and letting $r(\beta_t, x)$ "glide" along the boundary of $Q_i(\gamma_a)$ (Fig. 12).

The above observation is the basis for the algorithm to construct $\mathcal{F}_i^{tb}$. The general idea of the algorithm is as follows. Let us first consider a given γ. We can identify the set of (α, x)-values for which $vq_1(\gamma)$ is the last intersection point of $l(\alpha, x)$. What is more, it suffices for the algorithm to maintain the x-component of each (α, x); namely, for a given γ and x, only one α exists for which $l(\alpha, x)$ contains $vq_1(\gamma)$. We refer to the resulting set of x values as the *active subset* of X (Fig. 13a and b). Subsequently, the algorithm determines β_t for each x in the active subset. In summary, for a given γ, the algorithm models topple-unsafety by using $Q_i(\gamma)$ to identify the active subset of X and determine the corresponding β_t-values.

The above concept generalizes as follows. For an arbitrary initial γ_s, the algorithm constructs $Q_i(\gamma_s)$. Starting from γ_s, the algorithm then traverses Γ while maintaining $Q_i(\gamma)$. Simultaneously, it uses $Q_i(\gamma)$ to identify the active subset of X and the corresponding β_t-values for each visited γ. As a result, the algorithm constructs a planar subdivision [14] of the Γ, X-space. This subdivision specifies for each (γ, x) whether it is active and, if so, the corresponding β_t. Using the property that an active (γ, x) uniquely maps to a (α, x), the algorithm maps the Γ, X-space subdivision to the A, X-space, after which $\mathcal{G}_i^{tb}$ can be constructed using A, X-space subdivision and the stored β_t information.

In the following, we discuss the four main components of the algorithm in more detail. We start with a discussion on maintaining $Q_i(\gamma)$. Next, we

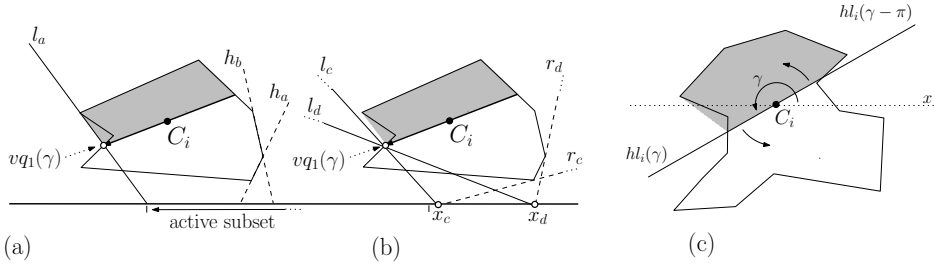

Figure 13: (a). An example of the identification of an active subset of X, which in this case goes to infinity for $+x$. Furthermore, h_a and h_b give an impression of how β_t-values will be linked to active x-values. (b). (α_c, x_c) and (α_d, x_d) are two examples of active (α, x) values for which β_t is defined by r_c and r_d. (c). $hl_i(\gamma)$ and $hl_i(\gamma - \pi)$ sweeping over Pi.

explain the mapping of $Q_i(\gamma)$ to the active subset of X, resulting in subdivision $\mathcal{S}_1$ of the Γ, X-space. We then continue by using $Q_i(\gamma)$ to identify the β_t-values, which further subdivides $\mathcal{S}_1$ and results in subdivision $\mathcal{S}_2$ of the Γ, X-space. Lastly, we map $\mathcal{S}_2$ from the Γ, X-space to the A, X-space where it defines $\mathcal{G}_i^{tb}$. We conclude by imposing the constraints that C_i must be over T and α is in $[\alpha_s, \alpha_e]$.

Let $hl_i(\gamma)$ denote the half-line extending from C_i such that its angle with the positive x-axis of the earlier introduced local coordinate system is γ. We start with $Q_i(\gamma_s)$ for γ_s. Intuitively, when increasing γ from $\gamma = \gamma_s$, the maintenance of $Q_i(\gamma)$ can be described as two parallel processes. On the one side, $hl_i(\gamma)$ sweeps over P_i and expands $Q_i(\gamma)$ based on the encountered points of P_i. On the other side, $hl_i(\gamma - \pi)$ removes points from $Q_i(\gamma)$ based on its sweep over P_i (Fig. 13(c)).

Algorithmically, it suffices to update $Q_i(\gamma)$ at a discrete number of γ-values. These updates occur when $hl_i(\gamma)$ encounter a vertex of P_i, or when $hl_i(\gamma)$ encounters a point of P_i such that a vertex is added to or removed from $Q_i(\gamma)$. We refer to updates related to $hl_i(\gamma)$ as *left events* (Fig. 14). Similarly, we refer to updates related to $hl_i(\gamma - \pi)$ as *right events*. For all other γ values, the change of $Q_i(\gamma)$ is limited to the predictable movement of the intersection points of $hl_i(\gamma)$ and $hl_i(\gamma - \pi)$ with a specific edge of P_i. We note that these moving intersection points correspond to $vq_1(\gamma)$ and $vq_m(\gamma)$, respectively.

The algorithm to maintain $Q_i(\gamma)$ is a relatively straightforward adaptation of the Graham scan [20]. Using a similar approach, the (counter-

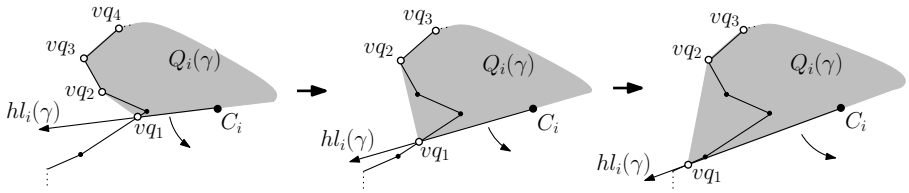

Figure 14: The three figures show three stages of a partial $Q_i(\gamma)$ in order of occurrence. The transitions between the three figures show two examples of left events: the disappearance of a $Q_i(\gamma)$ vertex, and $hl_i(\gamma)$ encounters a vertex of P_i.

clockwise) sweep of P_i by $hl_i(\gamma)$ effectively constructs the convex hull of P_i. On the other hand, the sweep with $hl_i(\gamma - \pi)$ effectively "deconstructs" the convex hull of P_i by inversely traversing the history of a similar, though clockwise construction of the convex hull of P_i. As a result, the following complexity and time bounds apply to the maintenance of $Q_i(\gamma)$ while traversing Γ.

Lemma 5. *The maintenance of $Q_i(\gamma)$ over all $\gamma \in \Gamma$ results in $O(n_i)$ left and right events and requires $O(n_i \log n_i)$ time.*

We conclude the discussion of maintaining $Q_i(\gamma)$ with a last note. We recall that an exception occurs when $vq_1(\gamma)$ is a vertex of an edge of P_i of which the supporting line contains C_i (Fig. 11a). Expressed in terms of the above $Q_i(\gamma)$ discussion, these exceptions occur for γ-values for which $hl_i(\gamma)$ contains such an edge of P_i. Consequently, for each of these γ-values, instead of one unique point, we need to consider all points of the edge contained in $hl_i(\gamma)$. We denote the set of these γ-values by Γ_{ex}, and we remark that Γ_{ex} contains $O(n_i)$ elements because P_i consists of $O(n_i)$ edges.

The mapping of $Q_i(\gamma)$ to $\mathcal{S}_1$ is as follows. For $\Gamma - \Gamma_{ex}$, we define a function $\xi_i(\gamma)$ that specifies the x-value of the intersection point of the half-line $\overrightarrow{vq_2(\gamma)vq_1(\gamma)}$ with the world x-axis, i.e. the track wall. Intuitively, the point $vq_1(\gamma)$ is the last intersection point between P_i and $l(\alpha, x)$ through $vq_1(\gamma)$ if and only if $x > \xi_i(\gamma)$ (Fig. 15). We note that this statement does not hold for $x > \xi_i(\gamma)$ for which C_i is not over T, but as mentioned above, these x values will be truncated in the final stage of the algorithm. Hence, for $\gamma \in \Gamma - \Gamma_{ex}$, the subdivision $\mathcal{S}_1$ is such that the interval $\langle \xi_i(\gamma), - \rangle$ is active. We refer to the set of active (γ, x) in $\mathcal{S}_1$ as the active subset of $\mathcal{S}_1$.

For $\gamma \in \Gamma_{ex}$, the algorithm momentarily fixes γ and lets $vq_1(\gamma)$ traverse

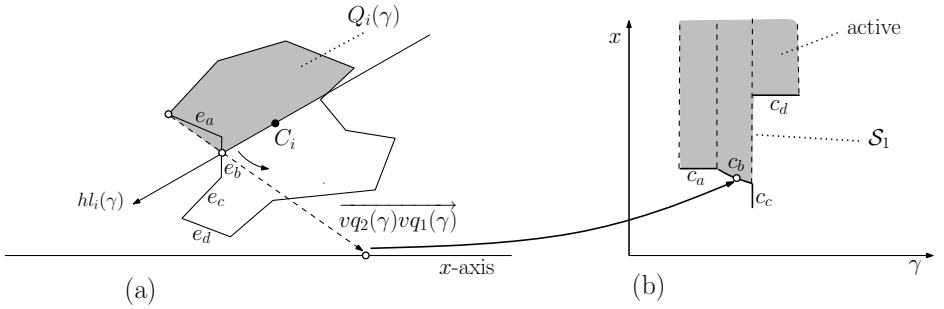

Figure 15: These figures illustrate a partial mapping from $Q_i(\gamma)$ to $\mathcal{S}_1$. The edges $e_a, \ldots, e_d$ map to curves $c_a, \ldots, c_d$ in the Γ, X-space. The gray area in the Γ, X-space is the set of active (α, x).

the edge contained in $hl_i(\gamma)$. The resulting x-values define one or more line segments parallel to the γ-axis, where the number of resulting line segments depends on the number of left events that change $vq_2(\gamma)$.

Lemma 6. *The planar subdivision $\mathcal{S}_1$ of the Γ, X-space is defined by $O(n_i)$ partial curves of constant complexity and requires $O(n_i)$ time to construct.*

Proof. Subdivision $\mathcal{S}_1$ is defined by two distinct sets of curve segments. The first set consists of line segments generated for Γ_{ex} and are clearly of constant complexity. Each $\gamma \in \Gamma_{ex}$ results in at least one line segment, while each additional line segment can be charged to a left event. The second set of curve segments defines $\xi(\gamma)$. Each such curve segment corresponds to a γ-interval for which $vq_2(\gamma)$ does not change and $vq_1(\gamma)$ moves along one specific edge of P_i. The latter condition breaks when $hl_i(\gamma)$ encounters a vertex of P_i, hence, each curve segment of $\xi(\gamma)$ can be charged to a left event. Furthermore, the curve segment is effectively described by a fixed point and edge, hence the complexity of the resulting curve is constant.

In summary, each curve segment can be charged to either one of the $O(n_i)$ elements of $\gamma \in \Gamma_{ex}$ or one of the $O(n_i)$ left event, thus, $\mathcal{S}_1$ consists of $O(n_i)$ partial curves. Each curve segment requires constant time to construct, hence the overall construction time is $O(n_i)$. □

Next, we add the β_t-information to our model by constructing $\mathcal{S}_2$. We observe that to identify β_t such that $r(\beta_t, x)$ is tangent to $Q_i(\gamma)$, it suffices to consider the vertices of $Q_i(\gamma)$; i.e., each $vq_i(\gamma)$ defines β_t for a set of active (γ, x) values. With this in mind, we construct $\mathcal{S}_2$ in two steps. Firstly, we use $Q_i(\gamma)$ to construct subdivision $\mathcal{S}_3$ of the Γ, X-space, where each $vq_i(\gamma)$

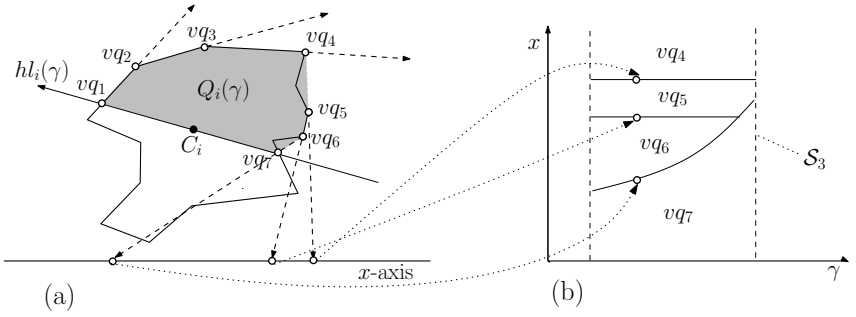

Figure 16: These figures illustrate a partial mapping from $Q_i(\gamma)$ to $\mathcal{S}_3$. The dashed arrows in (a) illustrate the half-lines defined by the $Q_i(\gamma)$ vertices. Three half-lines intersect the x-axis and map to curve segments in $\mathcal{S}_3$ in figure (b). Furthermore, for each cell of $\mathcal{S}_3$, figure (b) specifies the $Q_i(\gamma)$ vertex that will later be used to determine β_t.

maps to a cell of $\mathcal{S}_3$. A cell is a subset (of the Γ, X-space) defined by a set of curves. Secondly, we construct $\mathcal{S}_2$ by determining the intersection of $\mathcal{S}_3$ and the active subset of $\mathcal{S}_1$, which can be calculated using standard geometric techniques [14, 19].

For a given γ, the mapping of $Q_i(\gamma)$ to $\mathcal{S}_3$ is defined by the intersection points of the (world) x-axis with $\overrightarrow{vq_i(\gamma)vq_{i+1}(\gamma)}$ for each $i \in \{1, \ldots, m-1\}$ (Fig. 16). That is, for all x-values between two adjacent intersection points, β_t of tangent $r(\beta_t, x)$ is defined by one specific $vq_i(\gamma)$. Intuitively, the algorithm constructs $\mathcal{S}_3$ by traversing the Γ-domain while maintaining the intersection points; as a result, the above intersection points sweep out curve segments in the Γ, X-space, hence defining $\mathcal{S}_3$.

Lemma 7. *The planar subdivision $\mathcal{S}_2$ of the Γ, X-space is defined by $O(n_i^2)$ partial curves of constant complexity and requires $O(n_i^2)$ time to construct.*

Proof. Effectively, each curve segment in $\mathcal{S}_3$ corresponds to a pair of adjacent vertices of $Q_i(\gamma)$ and is defined for the γ-interval in which this pair exists. By using a proof similar to that of lemma 6, it can be shown that $\mathcal{S}_3$ consists of $O(n_i)$ curves of constant complexity and can be calculated in $O(n_i)$ time.

Hence, both $\mathcal{S}_3$ and the active subset of $\mathcal{S}_1$ are of complexity $O(n_i)$. Using standard geometric techniques, the intersection $\mathcal{S}_2$ of both subdivisions is of $O(n_i^2)$ complexity and require $O(n_i^2)$ to compute [14, 19]. □

The following discusses the mapping of $\mathcal{S}_2$ from the Γ, X-space to the

A, X-space. As discussed before, an active (γ, x) uniquely maps to a (α, x). We observe that the x-value remains unchanged, hence, this mapping effectively translates γ to α. Using basic geometric techniques, the algorithm maps each curve segment of $\mathcal{S}_2$ to a curve of the same complexity in the A, X-space. As will be shown below, this mapping is weakly order preserving; i.e., for any x, an arbitrary γ_1 and γ_2 with $\gamma_1 < \gamma_2$ map to α_1 and α_2 such that $\alpha_1 \geq \alpha_2$. Consequently, the mapping does not introduce new curve segment crossings, hence preserves the asymptotic complexity of $\mathcal{S}_2$. As a side note, we remark that previously disjunct curve segments may become (partially) overlapping.

The weakly order preserving property of the above mapping can be shown as follows. We recall that since the algorithm operates on $\mathcal{S}_2$, the mapping is only applied to active (γ, x) points. Let us consider an arbitrary active (γ_a, x_c) and (γ_b, x_c) such that $\gamma_a < \gamma_b$. Let $vq_a = vq_1(\gamma_a)$ and $vq_b = vq_1(\gamma_b)$ denote the last intersection points of $l(\alpha_a, x_c)$ and $l(\alpha_b, x_c)$. At this point, let us assume that $\alpha_a < \alpha_b$. From this assumption follows that vq_a is contained by the track adjacent to $l(\alpha_b, x_c)$. Subsequently, using the fact that P_i is represented by its star-hull, we can conclude that $\overline{C_i vq_a}$ intersects $l(\alpha_b, x_c)$. Let us denote the resulting intersection by i_{new}. Upon reflection, we can observe that since $\gamma_b > \gamma_a$ and $\gamma_a = \rho(i_{new})$, the distance of $(x_c, 0)$ to i_{new} is larger than to vq_b. This contradicts the fact that (γ_b, x_c) is active, hence $\alpha_a < \alpha_b$ can not be true.

The subdivision $\mathcal{S}_2$ mapped to the A, X-space defines $\mathcal{G}_i^{tb}$ as follows. The domain of $\mathcal{G}_i^{tb}$ is formed by the (active) cells of $\mathcal{S}_2$, i.e., each cell is a sub-domain of the domain of $\mathcal{G}_i^{tb}$. We will use the latter two concepts interchangeably. Recall that we only consider (α, x) such that C_i is over T and $\alpha \in [\alpha_s, \alpha_e]$. These two conditions define a subset of A, X of constant descriptive complexity. The algorithm truncates the domain of $\mathcal{G}_i^{tb}$ that does not intersect with this subset. Subsequently, using basic geometric techniques and the stored β_t-information, the algorithm generates $\mathcal{G}_i^{tb}$ by constructing a partial function for each remaining cell. In summary, $\mathcal{G}_i^{tb}$ is specified by a set of partial functions.

Recall that the topple function $\mathcal{F}_i^{tb}(\alpha)$ is defined as $\min\{\mathcal{G}_i^{tb}(\alpha, x) | x \in X_i(\alpha)\}$. This construction is realized in two steps. First, the algorithm projects each partial function of $\mathcal{G}_i^{tb}$ onto the A, B-plane, which results in a set of (boundary) curves for each projection. The resulting set of curves forms an arrangement [14] of curve segments in the A, B-plane. As a second step, the algorithm takes the lower envelope of this arrangement of curves, resulting in $\mathcal{F}_i^{tb}$. Lastly, we recall that $\mathcal{F}_i^{t}$ is a composition of $\mathcal{F}_i^{tb}$ and $\mathcal{F}_i^{ta}$. As

in previous function compositions, the function $\mathcal{F}_i^t$ is lower envelope of both functions. Lemma 8 shows that the complexity of $\mathcal{F}_i^t$ is almost quadratic.

Lemma 8. *The complexity of $\mathcal{F}_i^t$ is $O(n_i^2 2^{\alpha(n_i^2)})$ and the computation time is $O(n_i^2 2^{\alpha(n_i^2)})$, where $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function.*

Proof. Let us first consider the complexity of $\mathcal{G}_i^{tb}$. It has been shown that $\mathcal{S}_2$ consists of $O(n_i^2)$ curve segments, hence $\mathcal{S}_2$ can be subdivided in $O(n_i^2)$ cells of constant complexity. For a given cell, $\mathcal{G}_i^{tb}$ is based on a constant number of (fixed) vertices and edges of P_i; therefore, the complexity of the partial function for a given cell is constant. As a partial function and its domain description are both of constant complexity, clearly, its projection onto the A, B-plane results in a constant number of curve segments of constant complexity. In summary, the projection of the partial function for all cells in $\mathcal{S}_2$ results in an arrangement of $O(n_i^2)$ curve segments of constant complexity. The computation of each partial function and its subsequent projection requires constant time, hence the time cost so far is $O(n_i^2)$.

The construction of $\mathcal{F}_i^{ta}$ results in a similar arrangement of curve segments. We add both arrangements together, which does not change the asymptotic complexity. The function $\mathcal{F}_i^{tb}$ is the lower envelope of this last arrangement, which leads to an overall complexity and time cost of $O(n_i 2^{\alpha(n_i)})$ and $O(n_i 2^{\alpha(n_i)})$ [14, 19]. □

3.4 Solution

The composition of the trap function $\mathcal{F}_i^c$ by combining $\mathcal{F}_i^f$ and $\mathcal{F}_i^t$ is straightforward. Recall that all points on or below $\mathcal{F}_i^f$ and $\mathcal{F}_i^t$ correspond to trap designs that ensure P_i does not fall forward/backward nor toppling, respectively. Hence, intuitively, all points that are on or below both $\mathcal{F}_i^f$ and $\mathcal{F}_i^t$ correspond to safe traps for P_i. The function $\mathcal{F}_i^c$ is the lower envelope of $\mathcal{F}_i^f$ and $\mathcal{F}_i^t$.

Lemma 9. *$\mathcal{F}_i^c$ has a $O(n_i^2 2^{\alpha(n_i^2)})$ complexity and requires $O(n_i^2 2^{\alpha(n_i^2)})$ time to compute.*

Proof. Functions $\mathcal{F}_i^c$ and $\mathcal{F}_i^f$ are shown to be of $O(n_i^2 2^{\alpha(n_i^2)})$ and $O(n_i)$ complexity and require $O(n_i^2 2^{\alpha(n_i^2)})$ and $O(n_i)$ time to construct. The proof of the complexity and time costs of their composition is analogue to the proof of lemma 3. □

We generate $\mathcal{F}_i^c$ for each $i \in \{1, \ldots, k\}$ and add the result to the trap space, resulting in an arrangement of curves. We recall that a point that lies above all but one $\mathcal{F}_i^c$ in terms of β represents a valid solution; that is, it represents a V-trap that rejects all but one pose of P. The desired Σ is the subset of trap space between the upper envelope and the critical curves one level down, where the upper envelope is the curve comprised of all (partial) critical curves that are maximal with respect to β [14]. In geometry, this subset is referred to as the ≤ 1-*level* of an arrangement of curves [14] (Fig. 6).

For reference in the next section, we introduce the *solution function*, $\mathcal{F}^s$, which is defined by the aforementioned upper envelope. Note that this function effectively specifies the largest possible β for each α such that only one P_i survives $T(\alpha, \beta)$. For notational convenience, we use $T^s(\alpha)$ as a shorthand for $T(\alpha, \mathcal{F}^s(\alpha))$, and $P^s(\alpha)$ to denote the pose that is fed by $T^s(\alpha)$. Lastly, we remark that although $\mathcal{F}^s$ is a subset of the full solution set Σ, the function $\mathcal{F}^s$ does still specify all poses that can be fed by a V-trap.

Once Σ is generated, the algorithm reports the set of valid V-trap designs. Solution set Σ consists of a set of cells, each consisting of valid solutions that feed a specific pose of P. No valid solution exists when the ≤ 1-level is empty; in other words, when the entire upper envelope consists of coinciding critical curves. We conjecture that V-traps may not exist for certain classes of parts with prominent geometric symmetries; characterizing the class of parts for which V-traps are guaranteed to exist is a subject for future research.

Theorem 1. *The complexity of* Σ *is* $O(a2^{\alpha(a)})$ *and* Σ *can be constructed in* $O(a2^{\alpha(a)} \log a)$ *time, where* $a = n^3 2^{\alpha(n^3)}$.

Proof. By combining lemma 1 and lemma 9, we can conclude that complexity of the arrangement of critical functions is $\sum_{i=1}^{k} O(n_i^2 2^{\alpha(n_i^2)}) = O(n^3 2^{\alpha(n^3)})$ and, similarly, the time cost is $O(n^3 2^{\alpha(n^3)})$.

The complexity and computation time costs of the ≤ 1-level of an arrangement of n such curves is $O(n2^{\alpha(n)})$ and $O(n2^{\alpha(n)} \log(n))$, respectively [19]. Hence, the complexity and computation time costs of Σ is $O(a2^{\alpha(a)})$ and $O(a2^{\alpha(a)} \log(a))$, respectively, where $a = n^3 2^{\alpha(n^3)}$. □

4 Modeling Part Motion

So far, we have assumed that a part remains in contact with the track floor as it moves up the track. We recall, however, that part movement is caused by an asymmetric helical vibration. This means that in practice

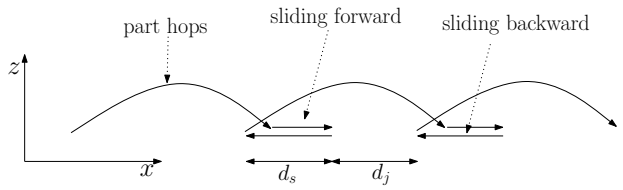

Figure 17: Illustration of a possible part motion modus, where d_s and d_j indicate the slide distance and the effective jump distance.

parts do not glide but jump along the track. Fig. 17 illustrates a motion modus proposed by Boothroyd [9], who applies this model among others in the context of trap design. The motion model can be explained as follows. In one vibration cycle, the upward motion of the bowl accelerates P, after which P is launched into flight as the bowl starts to move back. Next, P lands and slides forward a bit due to inertia, after which it slides back a bit again when the bowl returns to its upward motion. Depending on the part and bowl, different part motion modes are possible as well. However, all modes have in common that the part has some effective *jump distance*, d_j, and *slide distance*, d_s, where the latter may be 0. As a result, a P_i may unjustly survive $T(\alpha, \beta)$, in which case the trap does not satisfy the feeding property.

Given d_j and d_s, this section discusses an algorithm that takes as input the critical placement function $\mathcal{G}_i^{tb}(\alpha, x)$ for each P_i, and the solution function $\mathcal{F}^s$; taking the jumping motion into account, it outputs the set of all P_i that can be fed as well as the corresponding V-trap designs, or it reports that P can not be fed by a V-trap.

4.1 General Approach

In overview, our approach identifies the set of α-values for which, assuming the "worst case" jump pattern for each pose of P, trap $T^s(\alpha)$ still feeds $P^s(\alpha)$. The worst case jump pattern of a given pose P_i occurs when P_i jumps just before it reaches an unsafe placement, thus maximizing its chance to escape $T^s(\alpha)$. Let us refer to a α-value as *jump-unsafe* for a given pose P_i when $T^s(\alpha)$ still rejects P_i, independent of the jump pattern of P_i; let *jump-safe* be defined similarly. We note that a given α-value is per definition jump-safe for $P^s(\alpha)$.

Definition 6. *Trap $T^s(\alpha)$ feeds $P^s(\alpha)$ for a given α if and only if the given α is jump-unsafe for all poses of P (except $P^s(\alpha)$). We refer to such a α-value*

as well as the corresponding $T^s(\alpha)$ *as* jump-valid.

The identification of the set of jump-valid α-values is accomplished in two phases. First, we consider each P_i separately and determine the set of jump-unsafe α-values separately for forward- and backward-unsafety and both types of topple-unsafety; let the resulting sets be denoted by $\overline{A}_i^{ff}$, $\overline{A}_i^{fb}$, $\overline{A}_i^{ta}$ and $\overline{A}_i^{tb}$. The union of these four sets composes the overall set of jump-unsafe α-values for P_i, denoted by $\overline{A}_i$. So, for any $\alpha \in \overline{A}_i$, pose P_i will still be rejected by $T^s(\alpha)$ when taking jumping into account. For ease of explanation in the following, we mark the α-values for which $P_i = P^s(\alpha)$ also as elements of $\overline{A}_i$.

Second, we can determine the set of jump-valid α-values, denoted by $\mathcal{A}^s$, by computing the intersection of $\overline{A}_i$ over all poses of P; hence, formally defined: $\mathcal{A}^s = \bigcap_{i=1}^{k} \overline{A}_i$. With $\mathcal{F}^s$ and $\mathcal{A}^s$, we can report the set of poses of P that are still fed when taking jumping into account, as well as the trap designs to accomplish this. The following two subsections discuss the construction of $\overline{A}_i^{ff}$ and $\overline{A}_i^{tb}$—discussion of $\overline{A}_i^{fb}$ and $\overline{A}_i^{ta}$ is omitted for their construction is very similar to that of $\overline{A}_i^{ff}$ and $\overline{A}_i^{tb}$. The last subsection briefly addresses the composition of $\mathcal{A}^s$.

4.2 Jump-safety and Falling

We first consider a given α-value to illustrate the model, after which we discuss the realization of this concept using $\mathcal{F}^s$ and P_i. We finish with the identification of $\overline{A}_i^{ff}$.

Let us consider $\mathcal{F}^s$ for an arbitrary α. Intuitively, α is jump-safe (with respect to forward-unsafety) when P_i is able to jump from a placement in which only $l(\alpha, x)$ supports P_i to a placement in which $r(\mathcal{F}^s(\alpha), x)$ supports P_i. Clearly, the best chance to accomplish such a jump occurs, when the jump is executed in the placement just before P_i falls forward, i.e., when C_i is contained in $l(\alpha, x)$. Let us define $\xi_i^l(\alpha)$ such that $l(\alpha, \xi_i^l(\alpha))$ contains C_i. The above jump is successful when at least one point of P_i is contained in $r(\mathcal{F}^s(\alpha), x)$. Upon reflection, we can observe that the x-value closest $\xi_i^l(\alpha)$ that satisfies this condition is such that $r(\mathcal{F}^s(\alpha), x)$ is tangent to P_i (Fig. 18a). Let us define $\xi_i^r(\alpha)$ such that $r(\mathcal{F}^s(\alpha), \xi_i^r(\alpha))$ is tangent to P_i.

With the last safe placement to initiate the jump and the first safe placement to land, we now have both components to determine the x-distance that P_i needs to cross for a successful jump. If $\xi_i^l(\alpha) - \xi_i^r(\alpha) \leq d_j$, then P_i can jump across $T^s(\alpha)$ without falling forward; consequently, $\overline{A}_i^{ff} = \{\alpha \in [\alpha_s, \alpha_e] \,|\xi_i^l(\alpha) - \xi_i^r(\alpha) > d_j\}$.

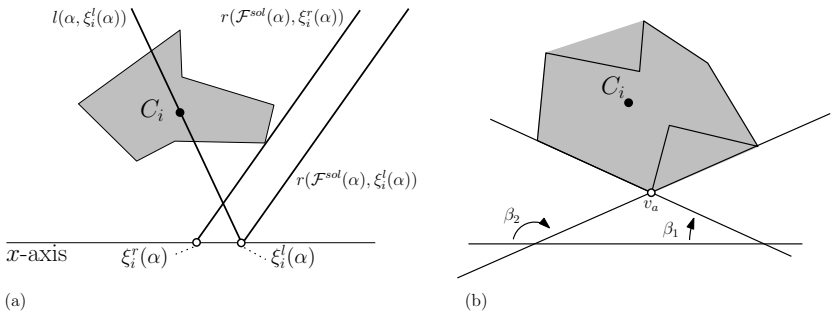

Figure 18: (a). $\xi_i^l(\alpha)$ defines $l(\alpha, \xi_i^l(\alpha))$ such that l contains C_i, and $\xi_i^r(\alpha)$ defines $r(\mathcal{F}^s(\alpha), \xi_i^r(\alpha))$ such that r is tangent to P_i. (b). all $r(\beta)$ for $\beta \in [\beta_1, \beta_2]$ intersect v_a when $r(\beta)$ is tangent to P_i.

The construction of $\xi_i^l(\alpha)$ is straightforward, hence let us consider $\xi_i^r(\alpha)$. The general idea is to map $\mathcal{F}^s$ from the A, B-space to the A, X-space based on the geometry of P_i. We start by observing that $r(\beta)$ tangent to P_i always touches a vertex of $\mathcal{CH}(P_i)$. Hence, conversely, a given vertex v of $\mathcal{CH}(P_i)$ translates to a β-interval for which $r(\beta)$ touches $\mathcal{CH}(P_i)$ in v (Fig. 18b). Intuitively, the β-interval corresponding to each vertex of $\mathcal{CH}(P_i)$ can be determined by rotating a line around $\mathcal{CH}(P_i)$, while maintaining the line tangent to $\mathcal{CH}(P_i)$. Subsequently, we split the B-range into the discovered β-intervals. This effectively subdivides the A, B-space into slabs spanning the entire A-domain for a given β-interval. Note that each slab corresponds to a specific vertex of $\mathcal{CH}(P_i)$ (Fig 19a).

We recall that $\mathcal{F}^s$ is defined by a set of curve segments of constant complexity. We examine each curve segment and, if needed, split it in sub-segments, such that each sub-segment is contained in one slab of the above subdivision. Lastly, we map each curve (sub)segment from the A, B-space to the A, X-space by using the $\mathcal{CH}(P_i)$ vertex of its containing slab. This concludes the construction of $\xi_i^r(\alpha)$.

The set $\overline{A}_i^{ff}$ can now be determined with a standard sweep approach. Intuitively, starting at α_s, we sweep a line $\alpha = \alpha_a$ through the A, X-space. As we increase α, we maintain the intersection points of $\xi_i^l(\alpha)$ and $\xi_i^r(\alpha)$ with the sweep-line, and add each α-interval for which $\xi_i^l(\alpha) - \xi_i^r(\alpha) > d_j$ to $\overline{A}_i^{ff}$ (Fig 19b).

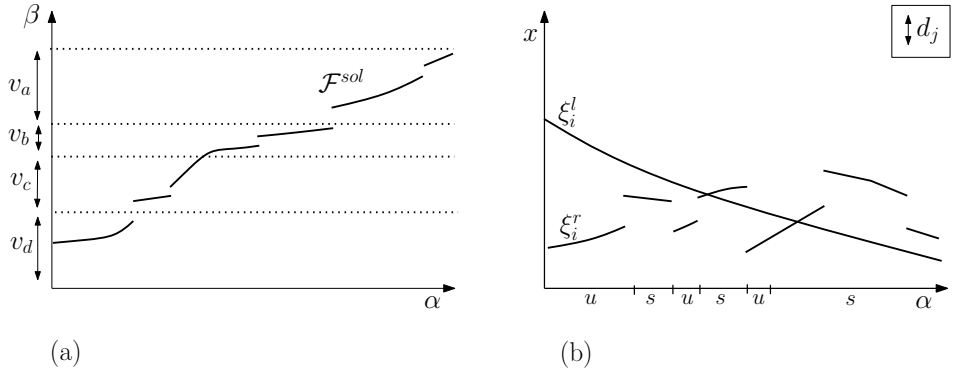

Figure 19: (a) $\mathcal{F}^s$, combined with the convex hull vertex information; each v corresponds to one slab. (b) $\xi_i^l(\alpha)$ and $\xi_i^r(\alpha)$ and the jump-safe (marked with 's') and jump-unsafe (marked with 'u') α-sets based on the given d_j.

4.3 Jump-safety and Toppling

For a given P_i, the jump-unsafe set $\overline{A}_i^{tb}$ is constructed by combining information of $\mathcal{F}^s$ and $\mathcal{G}_i^{tb}$. We first consider one α-value to illustrate our approach, then generalize the concept over $[\alpha_s, \alpha_e]$ and finally identify $\overline{A}_i^{tb}$.

Let us consider $\mathcal{F}^s$ and $\mathcal{G}_i^{tb}$ for an arbitrary α-value, denoted by α_a. We observe that $\mathcal{G}_i^{tb}(\alpha_a, x)$ is a function in the X,B-plane that specifies β_t for $x \in X_i(\alpha_a)$; intuitively, $\mathcal{G}_i^{tb}(\alpha_a, x)$ is a slice of the surface defined by $\mathcal{G}_i^{tb}$. Our first step is to identify the unsafe trap placements, i.e., we determine the x-intervals for which $T^s(\alpha_a)$ rejects P_i. We observe that these intervals consist of the x-values for which $\mathcal{G}_i^{tb}(\alpha_a, x) < \mathcal{F}^s(\alpha_a)$. Hence, the intervals can be identified by intersecting the line $\beta = \mathcal{F}^s(\alpha_a)$ with $\mathcal{G}_i^{tb}(\alpha_a, x)$. The product is a set of safe and unsafe x-intervals (Fig. 20).

An important observation is that we can discard safe intervals that are adjacent to two unsafe intervals and have a length shorter than d_s. This observation can be explained as follows. Let X_b be such a safe interval. Independent of where P_i lands in X_b after the jump over the preceding unsafe x-interval, the sliding motion still causes P_i to enter an unsafe placement and thus be rejected (Fig. 20). Consequently, we can merge safe x-intervals satisfying these criteria with their adjacent unsafe x-intervals, hence forming one larger unsafe x-interval. We will refer to this process as *eliminating false safety*.

Summarizing, for the given P_i and α_a, we have identified the (topple) unsafe trap placements of $T^s(\alpha_a)$, which is expressed as a set of x-intervals.

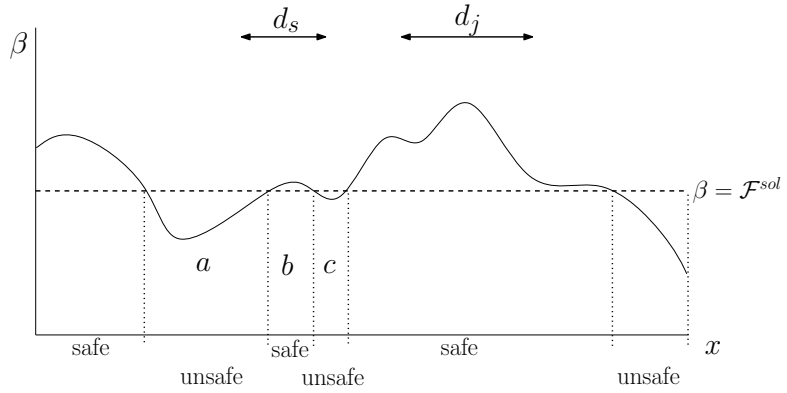

Figure 20: The figure shows an example slice of $\mathcal{G}_i^{tb}$ for a given α. In addition, it illustrates the identification of safe and unsafe x-intervals. The size of d_s and d_j is depicted with two line segments. In this example, safe x-interval b is shorter than d_s, hence will be merged with unsafe intervals a and c. The resulting new x-interval is larger than d_j, hence P_i can not jump over $T^s(\alpha)$.

Using this knowledge, we observe that P_i cannot jump over $T^s(\alpha_a)$ when d_j is smaller than the largest unsafe x-interval (Fig. 20).

In the following, we generalize the above discussed concept. Firstly, we define an expanded version of $\mathcal{F}^s$ in the placement space, which is defined as $\mathcal{G}^s(\alpha, x) = \mathcal{F}^s(\alpha)$. Intuitively, we expand $\mathcal{F}^s(\alpha)$ by extending a half-line parallel to the x-axis from $(\alpha, \mathcal{F}^s(\alpha), 0)$ for each $\alpha \in [\alpha_s, \alpha_e]$. Note that as $\mathcal{F}^s(\alpha)$ consists of constant degree curves, $\mathcal{G}^s(\alpha, x)$ consists of surface patches of constant complexity.

Our first step was the identification of safe and unsafe x-intervals, which here generalizes to safe and unsafe (finite) regions in the A, X-plane. These can be computed by first intersecting $\mathcal{G}^s(\alpha, x)$ with $\mathcal{G}_i^{tb}(\alpha, x)$ and projecting the result onto the A, X-plane. As both corresponding surfaces consists of constant degree patches, the resulting intersection curves are also of constant degree. The projection produces a planar subdivision [14, 19] of the A, X-plane consisting of constant degree curves, where each cell of the subdivision is marked safe or unsafe.

The second step is elimination of false safety by identifying (partial) false safe cells and merging them with their x-wise adjacent unsafe cells. The third and final step is to identify the α-intervals that contain a x-interval with a length larger than d_j, i.e., to extract $\overline{A}_i^{tb}$. Both steps can be executed in

one line sweep [14, 19]. That is, we sweep a line $\alpha = \alpha_a$ through the planar division. As we increase α, we maintain the one dimensional cross section of the above constructed planar division and report the desired α intervals. This sweep process is a complete process that takes a discrete amount of maintenance and report events, which depends on the complexity of the division. The set $\overline{A}_i^{tb}$ is the product of the sweep through $[\alpha_s, \alpha_e]$.

4.4 Combing Results

We recall that $\mathcal{A}^s$ is defined as $\bigcap_{i=1}^{k} \overline{A}_i$. First, we construct each $\overline{A}_i = \overline{A}_i^{ff} \cup \overline{A}_i^{fb} \cup \overline{A}_i^{ta} \cup \overline{A}_i^{tb}$ for all $i \in \{1, \ldots, k\}$. Next, we divide the resulting set of $\overline{A}_i$ in pairs, calculate their intersections, after which we repeat the former two steps. This process continues until either one partial solution is empty, or until all $\overline{A}_i$ are intersected and form $\mathcal{A}^s$. Combining $\mathcal{A}^s$ and $\mathcal{F}^s$, we report the set of feedable P_i and the trap designs to accomplish this.

5 Conclusion

We have presented an algorithm for the automated design of V-traps for vibratory bowl feeder—such devices receive a stream of identical polyhedral parts in arbitrary stable pose as input and output parts in one single pose. Inspired by similar devices used in existing vibratory feeder systems, the proposed work on V-traps serves as a representative study to feeding 3D parts through rejection of part poses. Under the assumption that the motion of the part is quasi-static and rejected parts are always deflected off the track, the presented complete algorithm reports either all possible single V-trap solutions or that no solution exists. The feedability of a desired part pose can be determined by considering the reported valid solutions. In addition, we have removed a common idealized assumption on the motion of the parts and replaced it by a more realistic model for part motion, which has been incorporated in our design algorithm.

Our aim in this paper has been to take the design of complete algorithms for trap design to the realm of three-dimensional parts. We have also shown that concepts and techniques from the field of computational geometry are powerful tools for the development of efficient, complete algorithms for the design of feeding devices of 3D parts.

A challenging open problem is to fully characterize the class of parts that are feedable with a single V-trap. Such a classification can either motivate or rule out the necessity of a study of more complex trap shapes and their

design algorithms. Along similar lines, an interesting question is what power (e.g. in terms of feed rate) a sequence of V-traps can add to the power of a single trap.

A different area of future research is the interaction between the part and trap. Both previous and the herein presented research on algorithmic trap design works with the assumptions that parts move in a quasi-static manner and rejected parts are always reflected off the track. No research has been done to date on the relaxations of these idealizations.

References

[1] P. Agarwal, A. Collins, and J. Harer. Minimum trap design. *IEEE ICRA*, pages 2243–2248, 2001.

[2] S. Akella, W. Huang, K. Lynch, and M. Mason. Parts feeding on a conveyor with a one joint robot. *Algorithmica*, 26(3):313–344, 2000.

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

[4] D. Berkowitz and J. Canny. Designing part feeders using dynamic simulation. *IEEE ICRA*, pages 1127–1132, 1996.

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

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

[7] R.-P. Berretty, M. Overmars, and A. van der Stappen. Orienting polyhedral parts by pushing. *Comput. Geom.*, 21(1-2):21–38, 2002.

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

[9] G. Boothroyd. *Automatic Assembly and Product Design.* Taylor & Francis Ltd, 2005.

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

[11] M. Brokowski, M. Peshkin, and K. Goldberg. Optimal curved fences for part alignment on a belt. *ASME Trans. of Mechanical Design*, 117:27–34, 1995.

[12] M. Caine. The design of shape interactions using motion constraints. *IEEE ICRA*, pages 366–371, 1994.

[13] A. Christiansen, A. Edwards, and C. Coello. Automated design of parts feeders using a genetic algorithm. *IEEE ICRA*, pages 846–851, 1996.

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

[15] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE J. of Robotics and Automation*, pages 367–279, 1988.

[16] O. Goemans, , K. Goldberg, and A. van der Stappen. Blades: A geometric primitive for feeding 3d parts on vibratory tracks. *IEEE ICRA*, pages 1730– 1736, 2006.

[17] O. Goemans, A. Levandowski, K. Goldberg, and A. van der Stappen. On the design of guillotine traps for vibratory bowl feeders. *IEEE CASE*, pages 79–86, 2005.

[18] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[19] J. Goodman and J. Rourke, editors. *Handbook of Discrete and Computational Geometry (Second Edition)*. 2004.

[20] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[21] M. Jakiela and J. Krishnasamy. Computer simulation of vibratory part feeding and assembly. 2^{nd} *Int. Conf. on Discrete Element Methods*, pages 403–411, 1993.

[22] 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. *Int. J. of Production Research*, 32(11):2513–2524, 1994.

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

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

[25] M. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[26] G. Maul and M. Thomas. A systems model and simulation of the vibratory bowl feeder. 2^{nd} *International Conference on Discrete Element Methods*, 16(5):309–314, 1997.

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

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

[29] D. Reznik and J. Canny. Universal part manipulation in the plane with a single horizontally vibrating plate. *Robotics, the algorithmic perspective*, pages 23–24, 1998.

[30] J. Selig and J. Dai. Dynamics of vibratory bowl feeders. *IEEE ICRA*, pages 3299–3304, 2005.

[31] R. Silversides, J. Dai, and L. Seneviratne. Force analysis of a vibratory bowl feeder for automatic assembly. *ASME: Journal of Mechanical Design*, 127(4):637–645, 2005.

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

[33] R. Zhang and K. Gupta. Automatic orienting of polyhedra through step devices. *IEEE ICRA*, pages 550–556, 1998.

[34] T. Zhang, G. Smith, R.-P. Berretty, K. Goldberg, and M. Overmars. The toppling graph: Designing pin sequences for part feeding. *IEEE ICRA*, pages 139–146, 2000.