Planning robotic manipulation strategies for workpieces that slide

M. A. Peshkin and A. C. Sanderson

IEEE Transactions on Robotics and Automation 4:5 (October 1988)

PLANNING ROBOTIC MANIPULATION STRATEGIES FOR SLIDING OBJECTS

M. A. Peshkin and **A. C. Sanderson***

Robotics Institute

Carnegie-Mellon University

Pittsburgh, Pennsylvania 15213

ABSTRACT: A *configuration map* is defined and computed, mapping all configurations of a part before an elementary manipulative operation to all possible outcomes. Configuration maps provide a basis for planning the operation sequences which occur in parts-feeder designs or in more general sensorless manipulation strategies for robots. Sequences of elementary operations are represented as matrix-products of configuration maps for the individual operations. Efficient methods for searching the space of all operations sequences are described.

As an example we consider a class of parts feeders based on a conveyor belt. Parts arrive on the belt in random initial orientations. By interacting with a series of stationary fences angled across the belt, the parts are aligned into a unique final orientation independent of their initial orientation. The planning problem is to create (given the shape of a part) a sequence of fences which will align that part. We demonstrate the automated design of such parts feeders.

KEYWORDS: Planning, manipulation, sliding, friction, parts feeder, alignment, robot.

This work was supported by a grant from Xerox Corporation, and by the Robotics Institute, Carnegie-Mellon University.

* Current address: Philips Laboratories, Briarcliff Manor, NY 10510

1. Introduction

Many robotic operations involve a part which is free to slide on a tabletop or a conveyor belt. Strategies have been suggested which take advantage of sliding friction between the part and the surface it slides on to facilitate accurate positioning of the part or reliable grasping of it. Examples of such strategies include the hinge-grasp strategy used by Paul [12] and analyzed by Mason [6]; a centering and aligning strategy analyzed by Brost [1]; a programmable parts-aligner based on sequential pushing of an object with a straight fence studied by Mani and Wilson [4]; and a system devised by Erdmann and Mason [2] in which a part dropped at random into a rectangular tray is aligned by a sequence of tipping motions of the tray.

The above examples reduce positional and orientational uncertainty without using sensing. In each case the geometry of the part is known, the initial position and orientation are unknown, and a model of the interaction of the part with a manipulation device is used to develop sensorless manipulation strategies. The above strategies base their models of interaction on the results of Mason [6], which provide partial information about the motion of a sliding object. In this paper we utilize our previous results [8] [9] [7] [10] [11] which provide complete bounds on the possible motions of a sliding object, as a basis for planning sensorless manipulation strategies. The use of exact motion bounds derived from physical models leads to more efficient manipulation strategies, and enables performance of tasks which would not otherwise be achievable without sensors.

The planning method introduced here is based on a mapping between bounded sets of part configurations, called a *configuration map*, which describes a single operation. The configuration map is a convenient tool for planning operation sequences using an appropriate search strategy. The application of this planning method to the automated design of a sequential fence parts feeder is described in section .

2. Physics of Sliding

In this section we summarize the results which are used to calculate bounds on the motion of a pushed sliding part [8] [9] [7] [10] [11].

A sliding object has three degrees of freedom. If we require the object to be in contact with another object (a pusher), the sliding object retains two degrees of freedom, which are most conveniently expressed as the coordinates of a point in the plane called the *center of rotation* (COR). Any infinitesimal motion of the object can be expressed as a pure rotation $\delta\theta$ about some COR.

The motion of a sliding object (and therefore the location of the COR) is substantially affected by the object's distribution of weight on the sliding surface. The distribution of weight depends on the location of any "bumps" on the underside of the object, or in the case of nominally flat surfaces may be affected dramatically by tiny deviations from flatness. Since we wish to determine the motion of any object, without knowing the distribution of weight, our goal is to find the locus of CORs under *all* possible weight distributions.

The coefficient of friction with the sliding surface (μ_s) does not affect the motion of the object if we use a simple Coulomb model of friction. It is also assumed that all motions are slow (the *quasistatic approximation*.) Quasistatic speeds are discussed by Mason [5] and by Peshkin [11]. Recent work by Wang [13] treats the high speed limit.

The object being pushed is assumed to be a disk with its center of mass (CM) at the center. Given another object of interest (e.g. a pentagon) we can consider a disk centered at the CM of the pentagon, big enough to enclose it. Since any weight distribution on the pentagon could also be a weight distribution on the disk, the COR locus of the disk must enclose the COR locus of the pentagon. The locus for the disk therefore provides bounds on the locus for the real object.

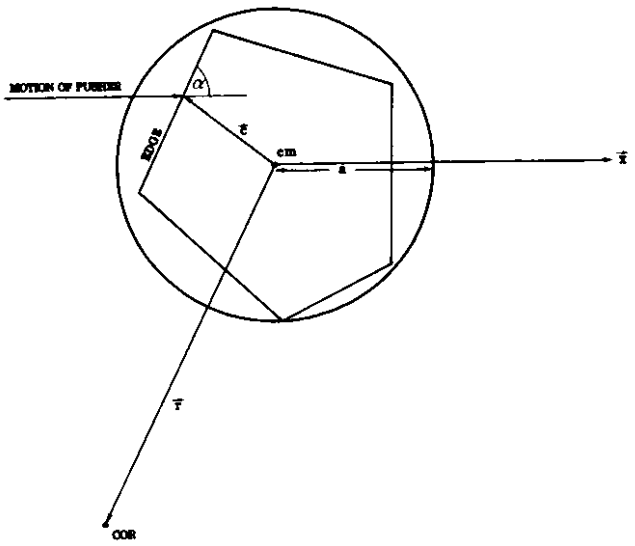

Figure 2-1: Parameters of the pushing problem

The parameters of the COR problem are the point of contact $\vec{c}$ between the pusher and the object, and the angle α between the edge and the line of pushing, as shown in figure 2-1. The values of α and $\vec{c}$ shown are the ones which are needed in considering the motion of the five-sided object shown inscribed in the disk. We do not require the point of contact $\vec{c}$ to be on the perimeter of the disk, as this would eliminate applicability of the results to objects inscribed in the disk. Similarly, we do not require α to be such that the edge being pushed is perpendicular to vector $\vec{c}$, as it would be if the object were truly a disk. The disk (with radius a), α, $\vec{c}$, and the CM, are shown in figure 2-1, along with what might be the COR for some particular distribution of weight.

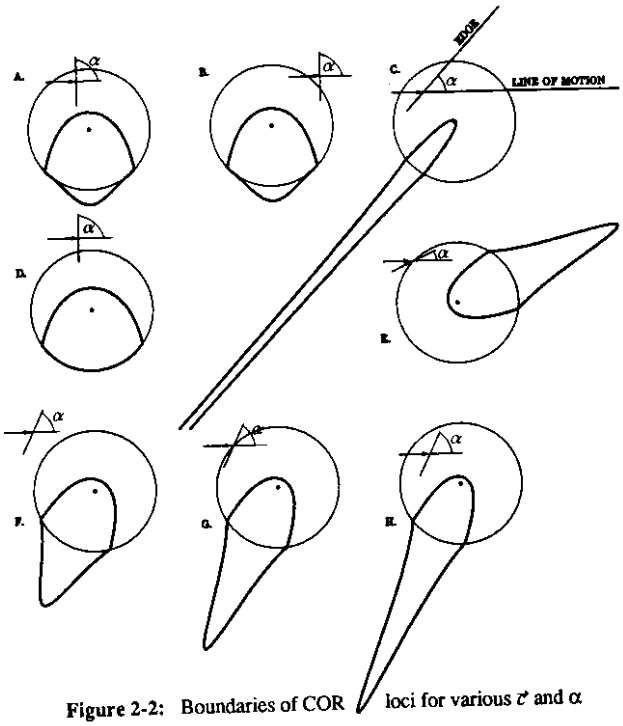

Figure 2-2: Boundaries of COR loci for various $\vec{c}$ and α

Figure 2-2 shows examples of the COR loci we found [8] [10] for various values of α and $\vec{c}$. In each section the angle α of the edge with respect to the line of pushing is indicated. The edge may be the edge of a pusher in contact with a corner of the inscribed object, or it may be an edge of the inscribed object in contact with a pushing point (as in figure 2-1). $\vec{c}$ is the vector from the CM (at the center of the disk) to the point of contact indicated by the arrowhead. The boundary of the COR locus is shown in bold outline. Every point within the locus is the COR for some possible distribution of weight on the disk. No distribution of weight can result in a COR outside the boundary shown. In figure 2-2, the coefficient of friction between the pusher and the object (μ_c) is zero. These elementary COR loci are denoted $\{COR\}_\alpha$.

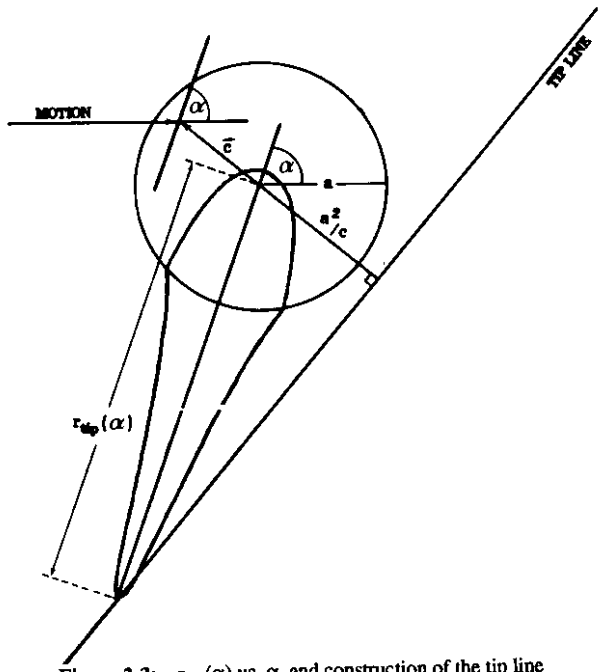

Figure 2-3: $r_{tip}(\alpha)$ vs. α, and construction of the tip line

Defining the unit vector $\vec{\alpha} = (\cos\alpha, \sin\alpha)$, we observe that the COR loci have an axis of symmetry about $\vec{\alpha}$. Note that the pushing force is directed perpendicular to $\vec{\alpha}$, (not parallel to the line of motion,) since $\mu_c = 0$. The distance r_{tip} from the CM to the farthest point of the COR locus is of particular usefulness, and we found [8] [10] [7] that it is

$$r_{tip} = \frac{a^2}{\vec{\alpha} \cdot \vec{c}} \tag{1}$$

As the angle α is varied, the tip of $\{COR\}_\alpha$ traces out a straight line called the *tip line*. The tip line, (figure 2-3), is perpendicular to $\vec{c}$, and a distance a^2/c from the CM.

If the coefficient of friction between pusher and object μ_c is non-zero, we found in [9] [7] that we can combine two of the elementary ($\mu_c = 0$) COR loci (such as are shown in figure 2-2), and the tip line construction, to create the COR locus comprising all the possible locations of the COR.

2.1. Application to interaction with a fence

As an example, consider a fence in linear motion which strikes and pushes an object. For a given initial orientation of the part, a particular point will be first struck by the fence. Whether a clockwise or a counterclockwise mode of rotation then occurs can be determined.

As the fence advances the part rotates, and it may also slip along the fence. The rates of rotation and slipping as the fence advances are bounded by the COR locus. The leftmost and rightmost points of the COR locus give the extremal slipping rates, and the highest and lowest points gives extremal rotation rate. The bounds allow calculation of the maximum distance the fence must advance to assure that an edge of the part has rotated into alignment, and the distance the part has slipped along the fence during alignment.

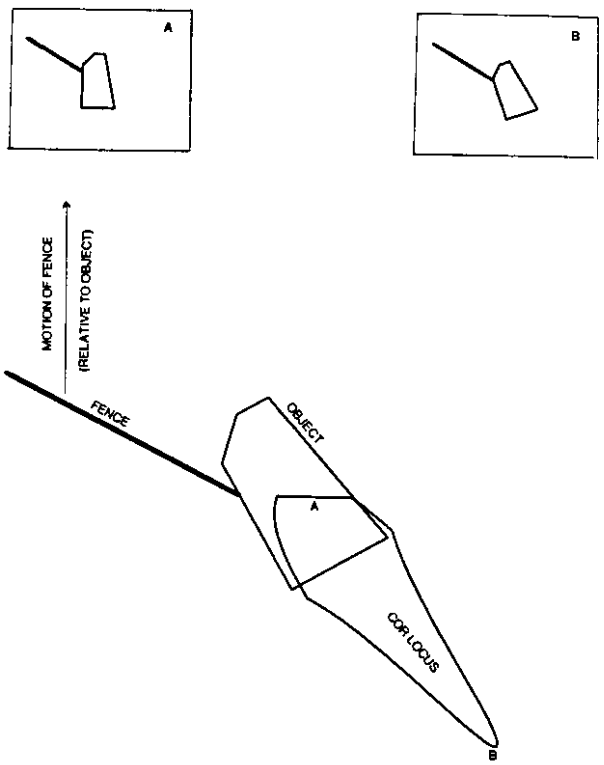

Figure 2-4: Extremal outcomes of interaction of part with of fence.

Finally, the part leaves the end of the fence (figure 2-4.) Two points of the COR locus (shaded) in the figure are of particular interest. If the COR is at point "A", the part will rotate without slipping relative to the fence. Rotation without slipping may persist until a face of the part is aligned with the motion of the belt, as shown in inset A. Point "B" gives another extreme of the possible motions of the part, in which the part slips relative to the fence as fast as possible for each increment of rotation. Maximal slipping may persist until the part loses contact with the fence, as shown in inset B. The extreme orientations shown in the two insets define the range of possible outcomes as the part interacts with the end of the fence. Point "A" and "B" have simple analytic forms. The motion of the part specified by point "B" can be integrated to find the extreme possible final orientation of the part shown in inset B.

3. Configuration Maps

The physics of an operation (for instance a collision between a fence and a part) may be encapsulated in a *configuration map*. A configuration map is a function of two copies of configuration space [3] (C-space × C-space), taking on logical values. A part lying on a tabletop has a three dimensional configuration space: it has two positional degrees of freedom and one rotational degree of freedom. The configuration map is therefore a function of six dimensions. Often, however, not all the degrees of freedom are of equal interest. For many purposes (e.g. planning a conveyor-belt based parts aligner), all we care about is the orientation of the part before and after its collision with a fence (or some other operation.) So while the configuration map representation is quite general, we will use here only a two-dimensional projection of it.

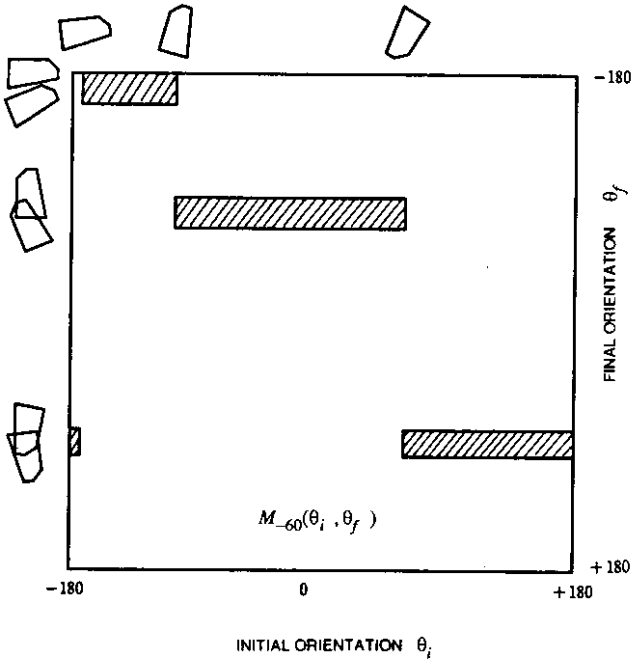

Figure 3-1: Configuration map for part interacting with –60 deg. fence.

Figure 3-1 shows the configuration map M_{-60} for the part and operation (interaction with a –60 degree fence) shown. We will consider the part to be on a moving surface traveling downward, but it could equally well be on a stationary surface with the fence moving upward, under robot control. The horizontal axis of the map gives the initial orientation θ_i of the part, before it contacts the fence. (Outlines of the part illustrate the orientations at several points along the axis.) The vertical axis gives the final orientation θ_f of the part after it has collided with the fence, rolled along the fence until a stable edge comes into contact with the fence, and finally slid down the fence and off the end. A point $M_{-60}(\theta_i, \theta_f)$ is shown shaded if it is nonzero (logical 1), where nonzero values indicate it is possible for a part with initial orientation θ_i to emerge with orientation θ_f from its interaction with the fence.

For any initial configuration of the part, the configuration map gives the final configuration. Note that in the case shown, for a single initial configuration there is a range of final configurations. This does not reflect a deficit in our physical understanding of the operation. The "one-to-many" mapping occurs because we are given only the outline of a part and do not know the distribution of the weight of the part upon the surface it slides on. The behavior of the part depends on the distribution of weight, which in turn depends on the generally unknown details of the surfaces in contact. Using our results [9] [7], the set of final orientations for all distributions of weight is calculated. The horizontal bands in this configuration map are associated with discrete alignments of the polygonal faces of the part.

The utility of the configuration map representation lies in the ease with which configuration maps for sequential operations can be calculated. In figure 3-2, a part is being carried along a belt, and will interact first with a –60 degree fence, and then with a +60 degree fence. A configuration map can be created which maps the part's initial configuration before colliding with the first fence, into its final configurations after leaving the second fence. That configuration map is simply the matrix product of the configuration maps for the two individual interactions. In the figure the two maps to be multiplied and their product are shown. The product M_{+60-60} is defined as

$$M_{+60-60}(\theta_i, \theta_f) = M_{+60} M_{-60}$$
$$= \vee_\alpha \{M_{+60}(\alpha, \theta_f) \wedge M_{-60}(\theta_i, \alpha)\}$$

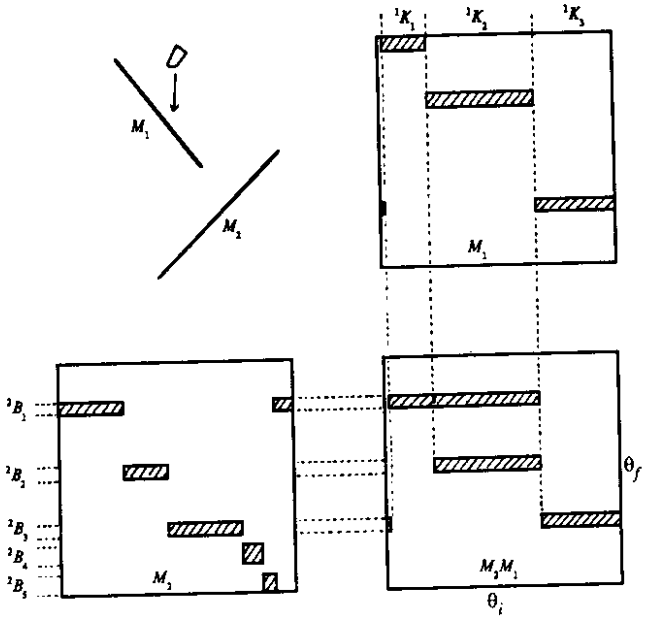

Figure 3-2: Product of two configuration maps.

3.1. Symbolic encoding

To take advantage of the "bands" evident in the configuration map, we construct N subintervals B_j of the θ_f axis, each bounding one of the bands. For each band B_j a kernel K_j of the θ_i axis is defined as

$$K_j = \cup_{\alpha \in B_j} \{\theta_i \mid M(\theta_i, \alpha) > 0\}$$

which is the set of initial configurations which lead to a final configuration in the band B_j. A new "rectangularized" map

$$M' = \cup_j \; K_j \times B_j$$

is nonzero wherever M is nonzero, and perhaps at other locations as well. When M is made up entirely of rectangular bands, as in figure 3-1, we have $M' = M$.

Now consider a product of two maps $M_2 M_1$ (operation M_1 followed by operation M_2). Using superscript 1 or 2 to indicate correspondence with one of the maps, we can express the product $M_2 M_1$ in terms of the bands of M_2 and the kernels of M_1:

$$M_2 M_1 = \cup_j \; {}^2B_j \times (\cup_{k \in {}^{21}C_j} {}^1K_k)$$
$$\text{where } {}^{21}C_j = \{k \mid {}^2K_j \cap {}^1B_k \neq \emptyset\}$$

The code sets ${}^{21}C_j$ contain all the information about the product. In the example shown in the figure, with the bands B_j as labeled, we have $C_1 = \{1,2\}$, $C_2 = \{2\}$, $C_3 = \{3\}$, $C_4 = C_5 = \emptyset$. (In figures, this code set would be written $1,2 \rightarrow 1$, $2 \rightarrow 2$, $3 \rightarrow 3$.) Further products can be computed using the code sets only, e.g. the code sets ${}^{321}C_j$ for the product $M_3 M_2 M_1$ are

$${}^{321}C_j = \cup_{k \in {}^{32}C_j} {}^{21}C_k$$

4. Planning Operations Sequences

The space of all operations sequences may be represented as a tree. Arcs correspond to operations, e.g. collisions with fences of various angles in our example. The root is labeled with the set of all initial configurations. Each node of the tree is labeled with the set of possible configurations of a part after execution of the operations on the path from the root to that node. In figure 4-1, part of a tree for operations which are collisions with fences of various angles is shown. The possible configurations of a part at a given node are obtained by multiplying the configuration maps for the operations on the path from the root to that node. The product maps for the six nodes shown along the left edge of figure 4-1 are shown in figure 5-2. Traversing the tree in order to search it is facilitated by the ease with which products of multiple configuration maps can be computed using the code sets C_j. In figure 4-1, each arc is labeled with a fence angle α as well as the code sets for that fence angle. The sets of possible configurations which label a node are indicated as a subset of the indices j of the bands ${}^{\alpha}B_j$ for the fence angle α of the arc above it.

A goal node is one in which the set of possible configurations has been reduced to one, or to a sufficiently narrow range. In figure 4-1 therefore, a goal node is labeled with just one band index j, such as is the depth 6 node on the left.

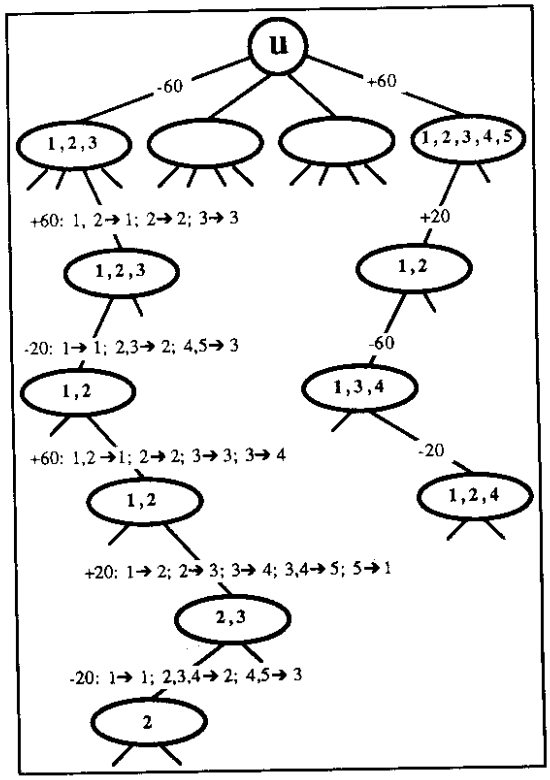

Figure 4-1: Tree for searching for an effective operations sequence.

4.1. Pruning the Tree

Searching the tree exhaustively (to any reasonable depth, e.g. six) is essentially impossible because the branching factor at each node is so high. Two techniques may be used to make the search practical.

Among the many arcs (fences) leaving a node, only a few distinct code sets will be observed. Suppose the arcs for fence angles +50 through +60 share code sets $1,2 \rightarrow 1$, $2 \rightarrow 2$, $3 \rightarrow 3$. It turns out (a result of the physics) that the final-orientation bands of a given fence are entirely contained in the corresponding final-orientation bands of a less steep fence. If a solution exists using the +55 degree arc from a given node, it must also exist using the +60 degree arc from that node. Given several arcs *having common code sets*, it is always safe to follow only the arc for the steepest fence.

The pruning step just described keeps the branching factor to a manageable level (typically 6-12). It is worth noting that for any particular incoming arc to a node, the collection of distinctly coded outgoing arcs can be precomputed.

Secondly, branches of the tree can be pruned while the tree is being searched. For each fence angle α, a list is kept of all node values computed after traversing an arc labeled α. If a node value is computed which is a superset of a previous value on the list, at a greater depth, the branch may be pruned. As an example, consider the depth four node labeled "1,2,4" in figure 4-1. It follows an arc for a fence angle of −20 degrees. A previously visited node of depth three labeled "1,2" also follows a −20 degree arc. "1,2,4" is a superset of "1,2". If a solution exists in n steps from "1,2,4", the same solution must also exist from "1,2". The total depth of the solution will be less starting from "1,2", so it is pointless to follow the "1,2,4" branch further.

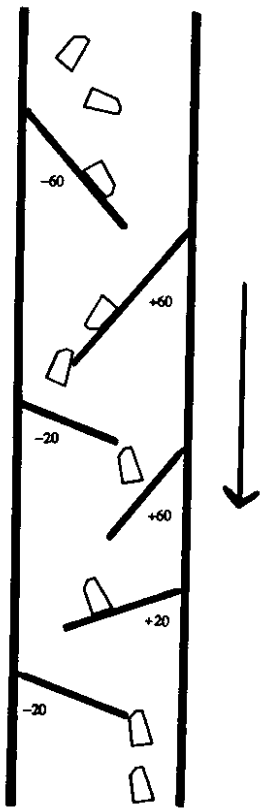

Figure 5-1: Top view of a parts-feeder design.

5. Example: Automated Design of a Parts-feeder

Figure 5-1 shows a top view of a system of fences suspended across a conveyor belt. The configuration map for the part shown with the first (−60 degree) fence was given in figure 3-1. The configuration map for the first two fences, considered as a unit, was given in figure 3-2. The configuration map for the entire system of six fences is shown in figure 5-2(f). Figure 5-2(a-e) are the partial products as labeled.

The configuration map for the system (figure 5-2(f)) has but one final-orientation band. Therefore the system of fences (figure 5-1) is a parts-feeder: parts in any initial orientation emerge from their interaction with the system of fences in but one range of final orientations. To reduce the range of final orientations to a single final orientation, the parts can be "used" (e.g. picked up by a robot) before they leave the final fence.

Some parts have two (or more) indistinguishable orientations. A rectangle, for instance, has two. For such parts the configuration map of a parts feeder has two (or more) final-orientation bands. A goal node of the search tree would be labeled with two (or more) band indices j.

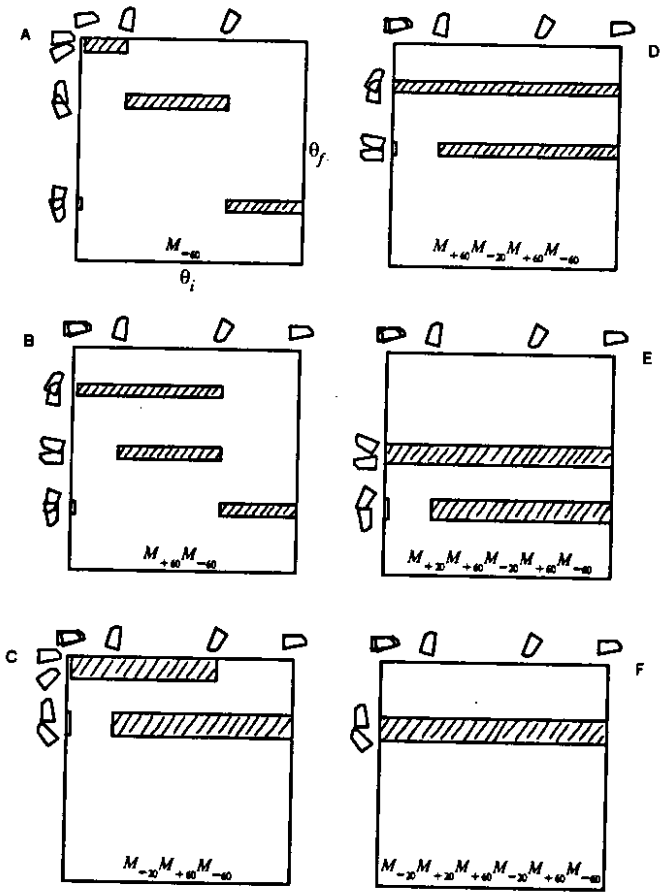

Figure 5-2: Configuration map products of successive fences.

Planning robotic manipulation strategies for workpieces that slide (pdf file)

M. A. Peshkin and A. C. Sanderson
IEEE Transactions on Robotics and Automation 4:5 (October 1988)

Abstract:

A configuration map is defined and computed, mapping all configurations of a part before an elementary manipulative operation to all possible outcomes. Configuration maps provide a basis for planning the operation sequences which occur in parts-feeder designs or in more general sensorless manipulation strategies for robots. Sequences of elementary operations are represented as matrix-products of configuration maps for the individual operations. Efficient methods for searching the space of all operations sequences are described.

As an example we consider a class of parts feeders based on a conveyor belt. Parts arrive on the belt in random initial orientations. By interacting with a series of stationary fences angled across the belt, the parts are aligned into a unique final orientation independent of their initial orientation. The planning problem is to create (given the shape of a part) a sequence of fences which will align that part. We demonstrate the automated design of such parts feeders.

Keywords:

Planning, manipulation, sliding, friction, parts feeder, alignment, robot.

Text of paper:

IEEE Transactions on Robotics and Automation 4:5 (October 1988) PLANNING ROBOTIC MANIPULATION STRATEGIES FOR SLIDING OBJECTS M. A. Peshkin A. C. Sanderson Robotics Institute Carnegie-Mellon University ABSTRACT A configuration map is defined and computed, mapping all configurations of a part before an elementary manipulative operation to all possible outcomes. Configuration maps provide a basis for planning the operation sequences which occur in parts-feeder designs or in more general sensorless manipulation strategies for robots. Sequences of elementary operations are represented as matrix-products of configuration maps for the individual operations. Efficient methods for searching the space of all operations sequences are described. As an example we consider a class of parts feeders based on a conveyor belt. Parts arrive on the belt in random initial orientations. By interacting with a series of stationary fences angled across the belt, the parts are aligned into a unique final orientation independent of their initial orientation. The planning problem is to create (given the shape of a part) a sequence of fences which will align that part. We demonstrate the automated design of such parts feeders. KEYWORDS Planning, manipulation, sliding, friction, parts feeder, alignment, robot. ACKNOWLEDGEMENTS This work was supported by a grant from Xerox Corporation, and by the Robotics Institute, Carnegie-Mellon University. INTRODUCTION Many robotic operations involve a part which is free to slide on a tabletop or a conveyor belt. Strategies have been suggested which take advantage of sliding friction between the part and the surface it slides on to facilitate accurate positioning of the part or reliable grasping of it. Examples of such strategies include the hinge-grasp strategy used by Paul @cite(pingle74) and analyzed by Mason @cite(mason86a); a centering and aligning strategy analyzed by Brost @cite(brost86); a programmable parts-aligner based on sequential pushing of an object with a straight fence studied by Mani and Wilson @cite(mani85); and a system devised by Erdmann and Mason @cite(erdmann86) in which a part dropped at random into a rectangular tray is aligned by a sequence of tipping motions of the tray. The above examples reduce positional and orientational uncertainty without using sensing. In each case the geometry of the part is known, the initial position and orientation are unknown, and a model of the interaction of the part with a manipulation device is used to develop sensorless manipulation strategies. The above strategies base their models of interaction on the results of Mason @cite(mason86a), which provide partial information about the motion of a sliding object. In this paper we utilize our previous results @cite(peshkin-cor1) @cite(peshkin-cor2) @cite(peshkin-aaai-86) @cite(peshkin-ieee-86) @cite(peshkin-phd) which provide complete bounds on the possible motions of a sliding object, as a basis for planning sensorless manipulation strategies. The use of exact motion bounds derived from physical models leads to more efficient manipulation strategies, and enables performance of tasks which

would not otherwise be achievable without sensors. The planning method introduced here is based on a mapping between bounded sets of part configurations, called a @i(configuration map), which describes a single operation. The configuration map is a convenient tool for planning operation sequences using an appropriate search strategy. The application of this planning method to the automated design of a sequential fence parts feeder is described in section @ref(example). @Section(Physics of Sliding) In this section we summarize the results which are used to calculate bounds on the motion of a pushed sliding part @cite(peshkin-cor1) @cite(peshkin-cor2) @cite(peshkin-aaai-86) @cite(peshkin-ieee-86) @cite(peshkin-phd). A sliding object has three degrees of freedom. If we require the object to be in contact with another object (a pusher), the sliding object retains two degrees of freedom, which are most conveniently expressed as the coordinates of a point in the plane called the @i(center of rotation) (COR). Any infinitesimal motion of the object can be expressed as a pure rotation @g(dq) about some COR. The motion of a sliding object (and therefore the location of the COR) is substantially affected by the object's distribution of weight on the sliding surface. The distribution of weight depends on the location of any "bumps" on the underside of the object, or in the case of nominally flat surfaces may be affected dramatically by tiny deviations from flatness. Since we wish to determine the motion of any object, without knowing the distribution of weight, our goal is to find the locus of CORs under @i(all) possible weight distributions. The coefficient of friction with the sliding surface (@g(m)@-(s)) does not affect the motion of the object if we use a simple Coulomb model of friction. It is also assumed that all motions are slow (the @i(quasistatic approximation.)) Quasistatic speeds are discussed by Mason @cite[Mason85] and by Peshkin @cite(peshkin-phd). Recent work by Wang @cite(wang86) treats the high speed limit. The object being pushed is assumed to be a disk with its center of mass (CM) at the center. Given another object of interest (e.g. a pentagon) we can consider a disk centered at the CM of the pentagon, big enough to enclose it. Since any weight distribution on the pentagon could also be a weight distribution on the disk, the COR locus of the disk must enclose the COR locus of the pentagon. The locus for the disk therefore provides bounds on the locus for the real object. The parameters of the COR problem are the point of contact @math[@vec(c)##] between the pusher and the object, and the angle @g(a) between the edge and the line of pushing, as shown in figure @ref(parameters). The values of @g(a) and @math[@vec(c)##] shown are the ones which are needed in considering the motion of the five-sided object shown inscribed in the disk. We do not require the point of contact @math[@vec(c)##] to be on the perimeter of the disk, as this would eliminate applicability of the results to objects inscribed in the disk. Similarly, we do not require @g(a) to be such that the edge being pushed is perpendicular to vector @math[@vec(c)##], as it would be if the object were truly a disk. The disk (with radius @math(a)), @g(a), @math[@vec(c)##], and the CM, are shown in figure @ref(parameters), along with what might be the COR for some particular distribution of weight. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Parameters of the pushing problem) @tag(parameters) @end(fullpagefigure) Figure @ref(various-cor-LOCI) shows examples of the COR loci we found @cite(peshkin-cor1) @cite(peshkin-ieee-86) for various values of @g(a) and @math[@vec(c)##]. In each section the angle @g(a) of the edge with respect to the line of pushing is indicated. The edge may be the edge of a pusher in contact with a corner of the inscribed object, or it may be an edge of the inscribed object in contact with a pushing point (as in figure @ref(parameters)). @math[@vec(c)##] is the vector from the CM (at the center of the disk) to the point of contact indicated by the arrowhead. The boundary of the COR locus is shown in bold outline. Every point within the locus is the COR for some possible distribution of weight on the disk. No distribution of weight can result in a COR outside the boundary shown. In figure @ref(various-cor-loci), the coefficient of friction between the pusher and the object @math[(@g(m)@-(c))] is zero. These elementary COR loci are denoted @math[{COR}@-(@g(a))]. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Boundaries of COR loci for various @math[@vec(c)] and @g(a)) @tag(various-cor-loci) @end(fullpagefigure) Defining the unit vector @math[@vec(@g(a)) ##=## (@cos @g(a),## @sin @g(a))], we observe that the COR loci have an axis of symmetry about @vec(@g(a)). Note that the pushing force is directed perpendicular to @vec(@g(a)), (not parallel to the line of motion,) since @math[@g(m)@-(c)#=#0]. The distance @math[r@-(tip)] from the CM to the farthest point of the COR locus is of particular usefulness, and we found @cite(peshkin-cor1) @cite(peshkin-ieee-86) @cite(peshkin-aaai-86) that it is @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) @tag(EP01)##r@-(tip)###=##@over(num "a#@+(2)", denom "@vec(@g(a)):@vec(c)##") @end(mathdisplay) @end(equation) As the angle @g(a) is varied, the tip of @math[{COR}@-(@g(a))] traces out a straight line called the @i(tip line). The tip line, (figure @ref(tip-line)), is perpendicular to @math[@vec(#c)], and a distance @math(a#@+(2)/c) from the CM. @begin(fullpagefigure) @blankspace(8.5 inches) @caption{@math[#r@-(tip)#](@g(a)) vs. @g(a), and construction of the tip line} @tag(tip-line) @end(fullpagefigure) If the coefficient of friction between pusher and object @math[@g(m)@-(c)] is non-zero, we found in @cite(peshkin-cor2) @cite(peshkin-aaai-86) that we can combine two of the elementary (@math[@g(m)@-(c)#=#0]) COR loci (such as are shown in figure @ref(various-cor-loci)), and the tip line construction, to create the COR locus comprising all the possible locations of the COR. @subsection(Application to interaction with a fence) As an example, consider a fence in linear motion which strikes and pushes an object. For a given initial orientation of the part, a particular point will be first struck by the fence. Whether a clockwise or a counterclockwise mode of rotation then occurs can be determined. As the fence advances the part rotates, and it may also slip along the fence. The rates of rotation and slipping as the fence advances are bounded by the COR locus. The leftmost and rightmost points of the COR locus give the extremal slipping rates, and the highest and lowest points gives extremal rotation rate. The bounds allow calculation of the maximum distance the fence must advance to assure that an edge of the part has rotated into alignment, and the distance the part has slipped along the fence during alignment. Finally, the part leaves the end of the fence (figure @ref(endpoint).) Two points of the COR locus (shaded) in the figure are of particular interest. If the COR is at point "A", the part will rotate without slipping relative to the fence. Rotation without slipping may persist until a face of the part is aligned with the motion of the belt, as shown in inset A. Point "B" gives another extreme of the possible motions of the part, in which the part slips relative to the fence as fast as possible for each increment of rotation. Maximal slipping may persist until the part loses contact with the fence, as shown in inset B. The extreme orientations shown in the two insets define the range of possible outcomes as the part interacts with the end of the fence. Point "A" and "B" have simple analytic forms. The motion of the part specified by point "B" can be integrated to find the extreme possible final orientation of the part shown in inset B. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Extremal outcomes of interaction of part with of fence.) @tag(ENDPOINT) @end(fullpagefigure) @section(Configuration Maps) The physics of an operation (for instance a collision between a fence and a part) may be encapsulated in a @i(configuration map). A configuration map is a function of two copies of configuration space @cite[Lozano-Perez83] (C-space @math(@mult) C-space), taking on logical values. A part lying on a tabletop has a three dimensional configuration space: it has two positional degrees of freedom and one rotational degree of freedom. The configuration map is therefore a function of six dimensions. Often, however, not all the degrees of freedom are of equal interest. For many purposes (e.g. planning a conveyor-belt based parts aligner), all we care about is the orientation of the part before and after its collision with a fence (or some other operation.) So while the configuration map representation is quite general, we will use here only a two-dimensional projection of it. Figure @ref(map) shows the configuration map @math(M@-(-60)) for the part and operation (interaction with a @math(-60) degree fence) shown. We will consider the part to be on a moving surface traveling downward, but it could equally well be on a stationary surface with the fence moving upward, under robot control. The horizontal axis of the map gives the initial orientation @math(@g(q)@-(i)###) of the part, before it contacts the fence. (Outlines of the part illustrate the orientations at several points along the axis.) The vertical axis gives the final orientation @math(@g(q)@-(f)###) of the part after it has collided with the fence, rolled along the fence until a stable edge comes into contact with the fence, and finally slid down the fence and off the end. A point @math(M@-(-60)(@g(q)@-(i)###,#@g(q)@-(f)####)) is shown shaded if it is nonzero (logical 1), where nonzero values indicate it is possible for a part with initial orientation @math(@g(q)@-(i)###) to emerge with orientation @math(@g(q)@-(f)###) from its interaction with the fence. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Configuration map for part interacting with @math(-60) deg. fence.) @tag(MAP) @end(fullpagefigure) For any initial configuration of the part, the configuration map gives the final configuration. Note that in the case shown,

for a single initial configuration there is a range of final configurations. This does not reflect a deficit in our physical understanding of the operation. The "one-to-many" mapping occurs because we are given only the outline of a part and do not know the distribution of the weight of the part upon the surface it slides on. The behavior of the part depends on the distribution of weight, which in turn depends on the generally unknown details of the surfaces in contact. Using our results @cite(peshkin-cor2) @cite(peshkin-aaai-86), the set of final orientations for all distributions of weight is calculated. The horizontal bands in this configuration map are associated with discrete alignments of the polygonal faces of the part. The utility of the configuration map representation lies in the ease with which configuration maps for sequential operations can be calculated. In figure @ref(product), a part is being carried along a belt, and will interact first with a @math(-60) degree fence, and then with a @math(+60) degree fence. A configuration map can be created which maps the part's initial configuration before colliding with the first fence, into its final configurations after leaving the second fence. That configuration map is simply the matrix product of the configuration maps for the two individual interactions. In the figure the two maps to be multiplied and their product are shown. The product @math(M@-(+60-60)) is defined as @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Product of two configuration maps.) @tag(PRODUCT) @end(fullpagefigure) @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) M@-(+60-60)(@g(q)@-(i)###, @g(q)@-(f)###) ##=## M@-(+60)#M@-(-60) ##=## @or@-(@g(a))### {M@-(+60)(@g(a)###,@g(q)@-(f)#) ###@and### M@-(-60)(#@g(q)@-(i)##,@g(a)#)} @end(mathdisplay) @end(equation) @subsection(Symbolic encoding) @label(symbolic-encoding) To take advantage of the "bands" evident in the configuration map, we construct @math(N) subintervals @math(B@-(j)) of the @math(@g(q)@-(f)###) axis, each bounding one of the bands. For each band @math(B@-(j)) a kernel @math(K@-(j)) of the @math(@g(q)@-(i)#) axis is defined as @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) K@-(j) ##=## @union@-(@g(a) @in B@-(j))### {@g(q)@-(i)###@vbar##M(@g(q)@-(i)##,@g(a)##)###>###0} @end(mathdisplay) @end(equation) which is the set of initial configurations which lead to a final configuration in the band @math(B@-(j)). A new "rectangularized" map @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) M#' ##=## @union@-(j)##### @math(K@-(j)##@mult##B@-(j)) @end(mathdisplay) @end(equation) is nonzero wherever @math(M) is nonzero, and perhaps at other locations as well. When @math(M) is made up entirely of rectangular bands, as in figure @ref(map), we have @math(M#'##=##M). Now consider a product of two maps @math(M@-(2)#M@-(1)) (operation @math(M@-(1)) followed by operation @math(M@-(2))). Using superscript 1 or 2 to indicate correspondence with one of the maps, we can express the product @math(M@-(2)#M@-(1)) in terms of the bands of @math(M@-(2)) and the kernels of @math(M@-(1)): @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) M@-(2)M@-(1) ##=## @union@-(j)######## @+(2)B@-(j) ##### @mult ##### {@union@-(k @in @+(21)C@-(j))##### @+(1)K@-(k)} @sr(where) ### @+(21)C@-(j) ##=## {#k##@vbar##### @+(2)K@-(j)### @inter ### @+(1)B@-(k) #@neq# @emptyset###} @end(mathdisplay) @end(equation) The code sets @math[@+(21)C@-(j)] contain all the information about the product. In the example shown in the figure, with the bands @math(B@-(j)) as labeled, we have @math[C@-(1)#=#{1,2},###C@-(2)#=#{2},###C@-(3)#=#{3},###C@-(4)#=#C@-(5)#=#@emptyset]. (In figures, this code set would be written @math(1,2@rightarrow 1,#######2@rightarrow 2,#######3@rightarrow 3).) Further products can be computed using the code sets only, e.g. the code sets @math[@+(321)C@-(j)] for the product @math[M@-(3)M@-(2)M@-(1)] are @blankspace(.1 inch) @begin(equation) @begin(mathdisplay) @+(321)C@-(j) #=# @union@-(k @in @+(32)C@-(j))###@+(21)C@-(k) @end(mathdisplay) @end(equation) @section(Planning Operations Sequences) The space of all operations sequences may be represented as a tree. Arcs correspond to operations, e.g. collisions with fences of various angles in our example. The root is labeled with the set of all initial configurations. Each node of the tree is labeled with the set of possible configurations of a part after execution of the operations on the path from the root to that node. In figure @ref(TREE), part of a tree for operations which are collisions with fences of various angles is shown. The possible configurations of a part at a given node are obtained by multiplying the configuration maps for the operations on the path from the root to that node. The product maps for the six nodes shown along the left edge of figure @ref(TREE) are shown in figure @ref(SIX-PRODUCTS). Traversing the tree in order to search it is facilitated by the ease with which products of multiple configuration maps can be computed using the code sets @math(C@-(j)). In figure @ref(TREE), each arc is labeled with a fence angle @g(a) as well as the code sets for that fence angle. The sets of possible configurations which label a node are indicated as a subset of the indices @math(j) of the bands @math(@+(@g(a))B@-(j)) for the fence angle @g(a) of the arc above it. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Tree for searching for an effective operations sequence.) @tag(TREE) @end(fullpagefigure) A goal node is one in which the set of possible configurations has been reduced to one, or to a sufficiently narrow range. In figure @ref(TREE) therefore, a goal node is labeled with just one band index @math(j), such as is the depth 6 node on the left. @subsection(Pruning the Tree) Searching the tree exhaustively (to any reasonable depth, e.g. six) is essentially impossible because the branching factor at each node is so high. Two techniques may be used to make the search practical. Among the many arcs (fences) leaving a node, only a few distinct code sets will be observed. Suppose the arcs for fence angles @math(+50) through @math(+60) share code sets @math(1,2@rightarrow 1,#######2@rightarrow 2,#######3@rightarrow 3). It turns out (a result of the physics) that the final-orientation bands of a given fence are entirely contained in the corresponding final-orientation bands of a less steep fence. If a solution exists using the @math(+55) degree arc from a given node, it must also exist using the @math(+60) degree arc from that node. Given several arcs @i(having common code sets), it is always safe to follow only the arc for the steepest fence. The pruning step just described keeps the branching factor to a manageable level (typically 6-12). It is worth noting that for any particular incoming arc to a node, the collection of distinctly coded outgoing arcs can be precomputed. Secondly, branches of the tree can be pruned while the tree is being searched. For each fence angle @g(a), a list is kept of all node values computed after traversing an arc labeled @g(a). If a node value is computed which is a superset of a previous value on the list, at a greater depth, the branch may be pruned. As an example, consider the depth four node labeled "1,2,4" in figure @ref(tree). It follows an arc for a fence angle of @math(-20) degrees. A previously visited node of depth three labeled "1,2" also follows a @math(-20) degree arc. "1,2,4" is a superset of "1,2". If a solution exists in @math(n) steps from "1,2,4", the same solution must also exist from "1,2". The total depth of the solution will be less starting from "1,2", so it is pointless to follow the "1,2,4" branch further. @section(Example: Automated Design of a Parts-feeder) @label(example) @label(planning) Figure @ref(feeder) shows a top view of a system of fences suspended across a conveyor belt. The configuration map for the part shown with the first (@math(-60) degree) fence was given in figure @ref(map). The configuration map for the first two fences, considered as a unit, was given in figure @ref(product). The configuration map for the entire system of six fences is shown in figure @ref(six-products)(f). Figure @ref(six-products)(a-e) are the partial products as labeled. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Top view of a parts-feeder design.) @tag(FEEDER) @end(fullpagefigure) @begin(fullpagefigure) @blankspace(8.5 inches) @caption(Configuration map products of successive fences.) @tag(SIX-PRODUCTS) @end(fullpagefigure) The configuration map for the system (figure @ref(six-products)(f)) has but one final-orientation band. Therefore the system of fences (figure @ref(feeder)) is a parts-feeder: parts in any initial orientation emerge from their interaction with the system of fences in but one range of final orientations. To reduce the range of final orientations to a single final orientation, the parts can be "used" (e.g. picked up by a robot) before they leave the final fence. Some parts have two (or more) indistinguishable orientations. A rectangle, for instance, has two. For such parts the configuration map of a parts feeder has two (or more) final-orientation bands. A goal node of the search tree would be labeled with two (or more) band indices @math(j). @subsection(Some Solutions) Figure @ref(SOLUTIONS) shows several parts and the lowest number of fences for which a parts feeder was found. In one case, no feeder design was found in a search to a depth of 20. Planning a feeder requires only a few seconds of computation. @begin(fullpagefigure) @blankspace(8.5 inches) @caption(No. of fences required in a feeder for each part shape) @tag(SOLUTIONS) @end(fullpagefigure)