The Design of Shape from Motion Constraints

by

Michael Edward Caine

Revised version of a thesis submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mechanical Engineering at the Massachusetts Institute of Technology on January 29 1993.

Abstract

The function performed by many objects can be expressed in terms of the constraints they impose on the motions of other objects. Cam shafts, gears, fixtures, wrenches and doorknobs are all examples of a class of objects whose shapes are designed to interact in ways that constrain their relative motion. This report examines an approach to the analysis and design of functional shape interactions represented as motion constraints. In this approach, a graphical representation for motion constraints is used as the basis for visualizing and reasoning about the function derived from shape. This representation also serves as an environment for the interactive design of functional shapes. Specifically, we utilize the configuration space representation to make explicit the motion constraints imposed by the shapes of interacting objects. We have developed a set of computational tools that permits these motion constraints to be displayed and directly manipulated by a designer in order to achieve desired functional properties. During this manipulation process, all motion constraint modifications are mapped back continuously into shape modifications to ensure the consistency between the constraints and shape. The representations and tools developed in the research have been applied to the visualization, analysis, and design of a set of orienting, fixturing and assembly devices for the automated assembly of planar parts.

Acknowledgments

I am incredibly fortunate to have had Tomás Lozano-Pérez as my thesis advisor. The inspiration for this work grew out of discussions with Tomás, and many of the key concepts and ideas presented in this thesis are due to him. I have benefited tremendously from his support, patience, and his uncanny ability to keep sight of the forest through the trees. Thank you!

I am grateful to Prof. Warren Seering, my committee chairman, for bringing me to the A. I. lab., and for giving me the freedom to explore unorthodox research topics. My other committee member, Dr. Kenneth Salisbury, provided a great deal of helpful advice on kinematics, approaches to research, and the construction of model gliders.

I would like to thank Prof. David Gossard for his comments on the presentation of this work, and to all the members of the MIT CAD Laboratory for their time, patience, and the use of their computing and video equipment. I would also like to thank my fellow 2.157 group members Andy Christian, Barb Hove and Jin Oh for helping to create the many wonderful tools which have found new life in parts of this research.

I owe a special debt of gratitude to Randy Brost for numerous insightful discussions on $2 + 1D$ configuration space, and for his encouraging comments during the early stages of this work.

Thanks to the many friends and colleagues at the M.I.T Artificial Intelligence Laboratory over the years. In particular I thank my closest peers Dave Brock, Brian Eberman, Sundar Narasimhan, Rajan Ramaswammy and Jose Robles for sharing the trials and tribulations of the thesis process, and especially to Dr. Dave for the race to finish. I am particularly indebted to "The Captain", Sundar Narasimhan, for his help in debugging the cspace code, for bringing the trace representation to my attention, and for the many discussions on design, CAD and robotics.

Special thanks go to my officemate, *Dr.* Erik Vaaler, for his humor, unique views on life, sharing late-night meals at the lab, and his keen advice on the road to doctor-dom. Thanks to my fellow members of Prof. Seering's research group, past and present: Brian Avery, Mike Benjamin, Ken Chang, Andy Christian, Steve Eppinger, Steve Gordon, Jim Hyde, Sarath Krishnaswammy, Peter Meckl, Ken Pasch, Rob Podoloff, Whit Rappole, Lukas Ruecker, Neil Singer, Bill Singhose, Kamala Sundaram, Bruce Thompson, Tim Tuttle, Karl Ulrich, Erik Vaaler, and Al Ward.

Many friends outside of the lab contributed to making my stay at MIT more enjoyable and enlightening: Shannon Nakaya, Deb Savage, Misa Kishi, Eugene Kaji, Bea and Dan Kleppner, ... and the many other friends from JAMS whose names can't be written with the available TeX fonts.

To my family, Steve, Brian, Pam, Liam and Allison, for their love, support and for providing much needed distraction in Maine.

To Kiyoko, for her love, support, and for just putting up with me.

And finally, to the memory of my parents, Phillip and Helen.

This research was performed at the MIT Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038, and in part by the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685.

Contents

Introduction

Chapter 1

This report presents a set of representations, methodologies and tools for the purpose of visualizing, analyzing and designing functional shapes in terms of constraints on motion. The core of the research is an interactive computational environment that provides an explicit visual representation of motion constraints produced by shape interactions, and a series of tools that allow for the manipulation of motion constraints and their underlying shapes for the purpose of design.

1.1 Form and Function

What function does shape serve? More specifically, how do we define object shapes that are considered to be useful or functional? We can classify the function that shapes perform based on a number of possible criteria, a few of which might include the following:

- **Contacting shapes** include surfaces brought into contact to constrain the relative motions of objects. Automotive cam shafts, worm gears, vises, robot grippers, wrenches and bolt heads are all examples of objects whose shapes were carefully designed to constrain motion. Many common everyday objects such as tables, door handles, telephone handsets, car steering wheels and coffee cups also derive their function in some way or another by constraining the motions of other objects with which they come into contact.

- **Structural shapes** include objects that connect points in space, or bridge the space between contacting shape surfaces. Coupler links, connecting rods, I-beams, table legs and eyeglass frames are all examples of objects that serve to support contacting shape surfaces and whose shapes are designed to maximize some properties, such as strength and stiffness, while minimizing others, such as weight and cost.

- **Enclosing shapes** describe objects that span regions of space in order to cover or enclose that region. Roofs, walls, automobile windshields and front hoods

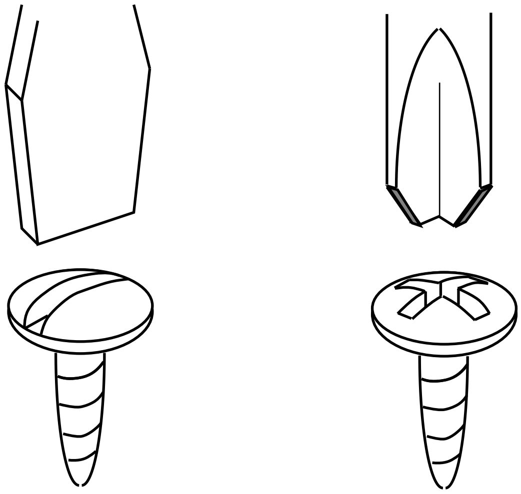


Figure 1.1: Slotted and phillips head screws and screwdrivers.

are a few examples of shapes that are designed to separate regions of space.

- **Aesthetic shapes** including such things as sculptures and car bodies that are designed primarily to satisfy given aesthetic criteria.

We note that various features or aspects of a single object may span some or all of the above categories. This is not surprising since objects are often designed to perform a number of functions simultaneously. In this report we will primarily concern ourselves with the class of contacting shapes that are designed to constrain motion.

A Familiar Example

Figure 1.1 shows two common fastener - driver shape pairs: a slotted screw and screwdriver, and a phillips head screw and screwdriver. Clearly, the shapes of the driver and screw head are important in both cases to the function of coupling with, and applying forces and torques to, the screws for their insertion or removal. Figure 1.2 shows the same two fasteners and drivers, but with the pairings between fasteners and drivers reversed. Our intuition and experience tells us that in this situation the once-functional screwdriver shapes are no longer useful for inserting or removing the screws. Why?

A Less Familiar Example

Figure 1.3 shows three vibratory part feeder geometries used to orient small parts. As the parts move through a feeder and interact with its features under vibratory

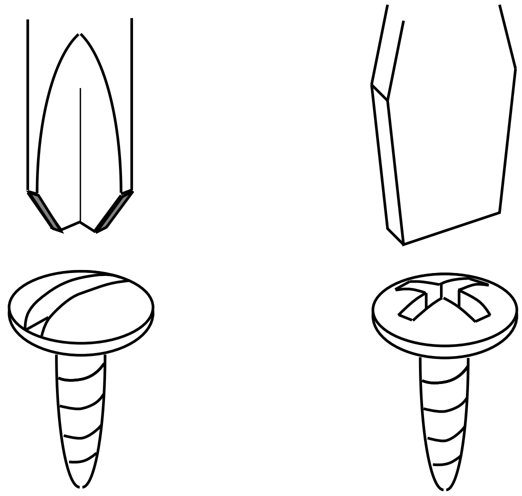


Figure 1.2: Slotted and phillips head screws paired with phillips and slotted screw-drivers.

motion, parts in all but one desired orientation fall out of the feeder to the side, while those parts in the desired orientation are allowed to pass through to be picked up by a robot.[1] The interesting thing to note about the three feeders in Figure 1.3 is that the geometrically similar tracks in (**a**) and (**b**) are functionally quite different, whereas the dissimilar tracks in (**a**) and (**c**) are functionally equivalent. Specifically, the feeder track shown in (**a**) outputs the el-shaped part shown in only one orientation, whereas all other orientations of the part will be knocked off the track and fall back into the bowl. The track shown in (**b**), however, although only slightly different from track (**a**) will output parts in **two** possible orientations and is therefore unacceptable for automated assembly. Finally, the feeder track shown in Figure 1.3 (**c**) will only output parts in the same orientation as track (**a**), and hence is functionally equivalent to (**a**).

In the first two example feeders (**a**) and (**b**), one pairing of geometries performs a useful function, while another pairing of apparently similar geometries fails to perform the same intended function. Similarly, in the second and third two example feeders (**b**) and (**c**), quite dissimilar geometries perform the same function. Why?

More specifically:

- Why does one pairing of shapes exhibit the desired functional characteristics while the other pairing does not?

- What are the important characteristics that determine the functionality of a

[1]The detailed characteristics of feeder construction and operation are given in Section 3.2.

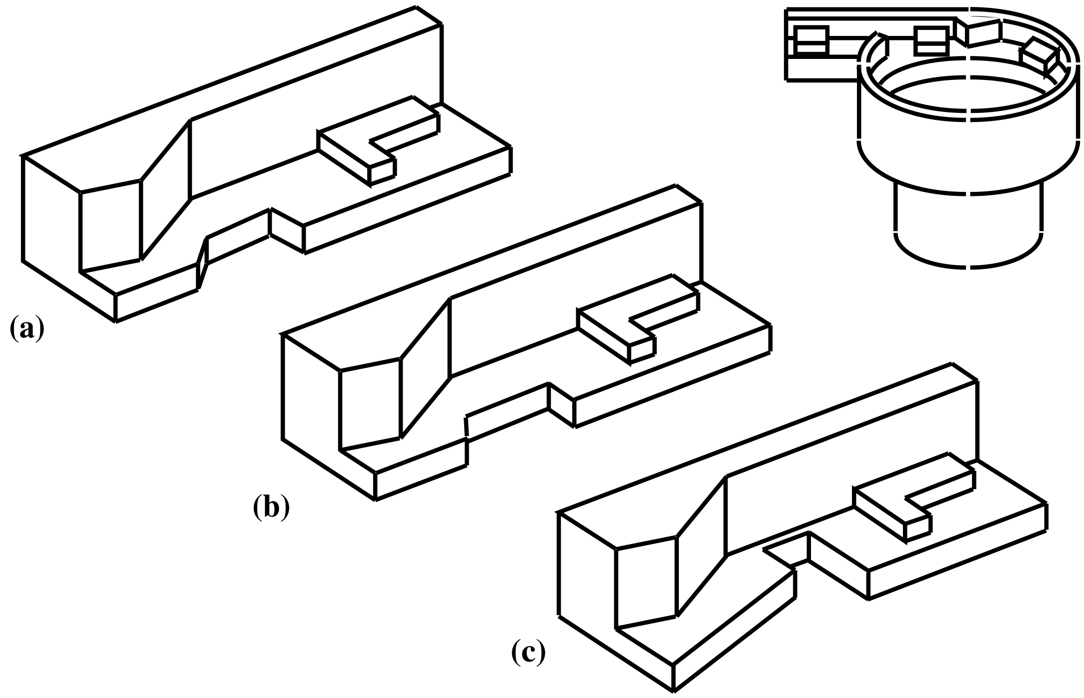


Figure 1.3: Three vibratory parts feeder geometries designed to orient the parts shown. Under given vibratory motion, feeder (**a**) succeeds in orienting the parts, whereas the geometrically similar feeder in (**b**) does not. The very dissimilar feeder geometry in (**c**), possesses the same feeding function as feeder (**a**).

given set of shapes?

- How would one go about designing shapes, or modifying an existing shape, to perform a desired function?

This report will address these and related questions.

The previous examples were chosen to highlight a number of important points about shape and function. First, the fastener example illustrates the observation that the functionality of object shapes is derived from shape *interactions* and not from individual shapes alone. Functional shapes become essentially useless when they were used outside the context of their intended interaction with other functional shapes. Second, the feeder example illustrates that our intuition about the function of shape depends to a large extent on our degree of familiarity with the problem domain. Like the fastener–driver example in Figures 1.1 and 1.2, the shapes of the feeders and the parts are clearly important to the function of orienting the parts. However, the precise role that the various shapes play in the stated function is less clear since the domain is less familiar. Because our intuition about the functional role of shape interactions is rather brittle outside of simple and familiar domains, we are less able to make appropriate design decisions. As a result, a task such as vibratory feeder design requires a considerable amount of trial and error, and is considered something of a “black art”.

1.2 Shape and Motion Constraints

Let us focus our discussion of *function* derived from *form* in the previous section to one of *motion constraints* derived from *shape*. Recall the vibratory feeders shown in Figure 1.3. Although sufficient for a basic understanding of feeder function, the brief description given of how a vibratory bowl feeder works doesn't tell us *(a)* if a particular feeder example will work, or *(b)* how to go about designing a feeder. Clearly we need a more precise description of the constraints imposed by interactions between shapes – we need a model of motion constraints.

Consider again the screw and driver examples from Figures 1.1 and 1.2. Let us select a set of parameters that describe the location of the screwdriver relative to the screw head. In the simplified case we may assume that the screw and driver are coaxially aligned so that only the distance between them along their common axis and the twist of the driver relative to the screw head are variable. These two variables, labeled Z and Ω respectively, are illustrated in Figure 1.4. If we plot the range of values for distance Z and twist Ω where the driver and screw head are not overlapping, we end up with a plot as shown in the right hand of the figure where the shaded region represents *occluded* placements of the driver relative to the screw head. Free regions in this two-dimensional space represent allowable placements of

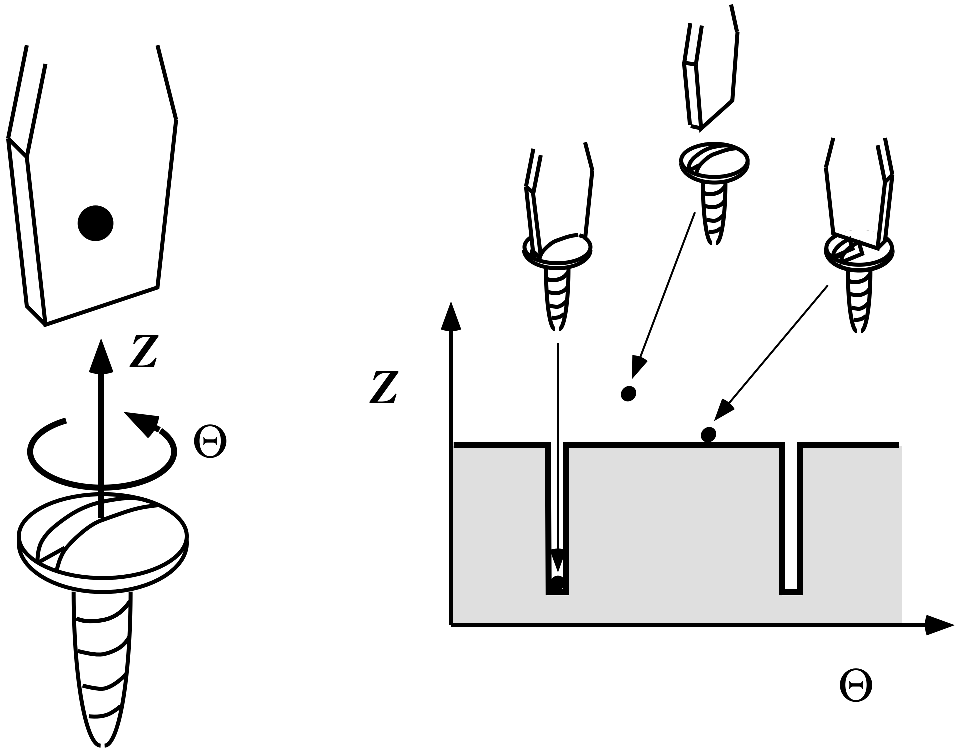


Figure 1.4: A representation of motion constraints for the slotted screw and screw-driver pair.

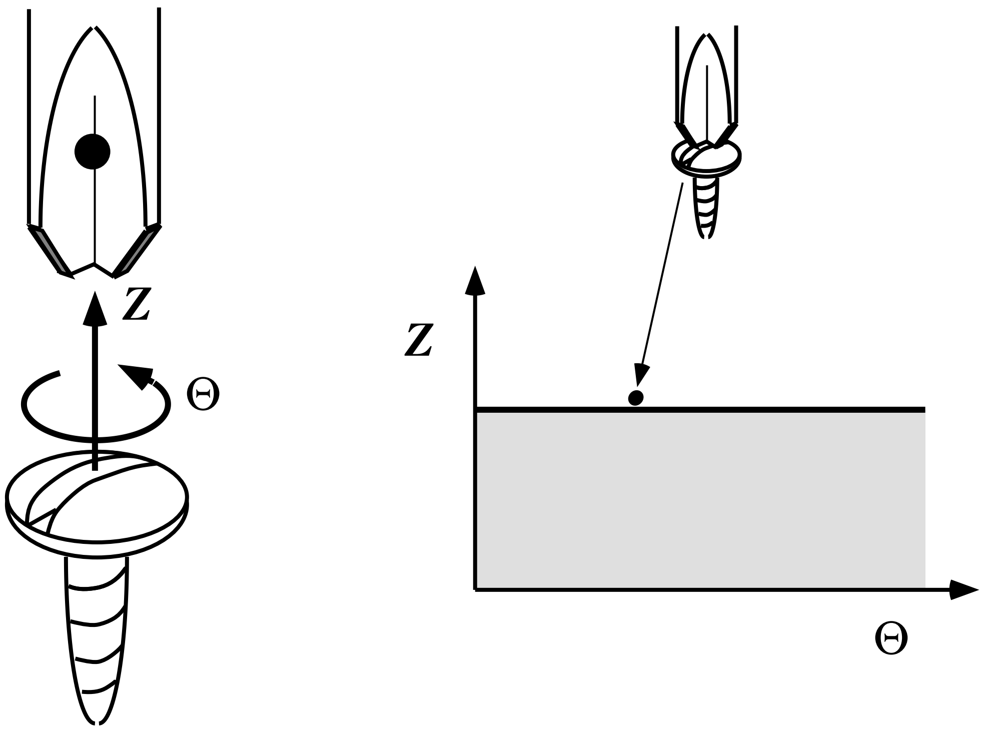


Figure 1.5: A representation of motion constraints for the slotted screw and phillips screwdriver pair.

the driver relative to the screw. The boundary separating the free and occluded regions represents placements of the driver that are in contact with the screw.

We notice a notch in the occluded region of Figure 1.5 corresponding to placements where the blade of the driver is in the screw head's slot. If we consider the function of the screwdriver to be one of transmitting torque to the screw head, then we can represent this function in terms of the motion constraints between the driver and screw head as captured by the contact boundaries in Figure 1.4. Specifically, if we consider the driver to be represented as the location of a selected point on the driver in the two-dimensional (Z, Ω) motion constraint diagram of Figure 1.4, then the only way to twist the screw head is to "push" against one of the two vertical sides of the notch in the Ω direction. Pushing in the $-\Omega$ direction corresponds to applying a clockwise torque to the screw, and pushing in the $+\Omega$ direction is equivalent to a counter-clockwise torque applied to the screw.[2]

The capacity of the above representation to capture the function of the screw–driver interaction is further illustrated by the motion constraint diagram for the slotted screw–phillips driver pairing shown in Figure 1.5. Here the characteristic notch, whose sides provided the constraint surfaces on which the driver could act on the screw, is missing. As expected, this screw–driver pairing does not exhibit the desired functionality of allowing torque to be transmitted from the driver to the screw.

In addition to extracting the function inherent in existing shape interactions, we are also interested in *generating* contacting shapes with desired functional characteristics. For example, in the case of the slotted screw–driver interaction of Figure 1.4, assume that we wish to be able to drive the fastener into a workpiece, but that we don't want the fastener to be removable. Figure 1.6 shows a motion constraint diagram with the desired properties, where the right side of the constraining notch has been sloped as shown so that the (Z, Ω) point representing the driver will slide along the constraint boundary and out of the notch rather than allow a torque to be applied to the screw. The corresponding "one-way" screw head shape shown might be generated from the new motion constraints by, for example, sweeping the screwdriver blade along the boundary of the motion diagram. This proposed shape synthesis process is complicated by the fact that inverting shapes is not unique. For example, rather than using the motion constraints in Figure 1.6 to produce a one-way screw, we could just have easily ended up with a one-way screwdriver, as shown in Figure 1.7.

[2]We are treating the relationship between forces and torques somewhat superficially at this level. Section 2.4.1 will address these issues in more detail.

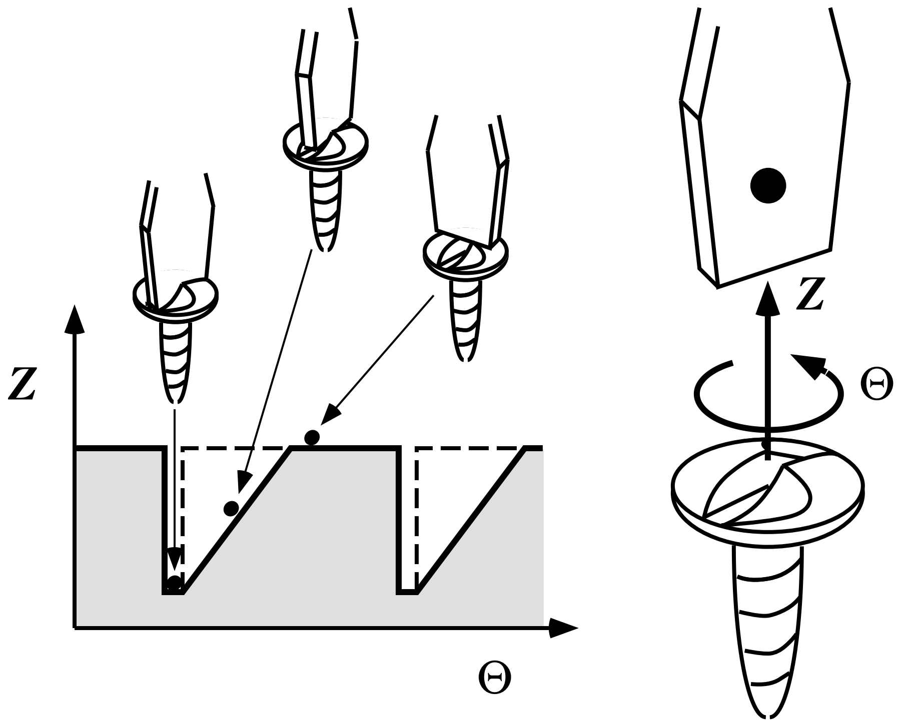


Figure 1.6: The "one way screw", one possible instantiation of a modified set of motion constraints.

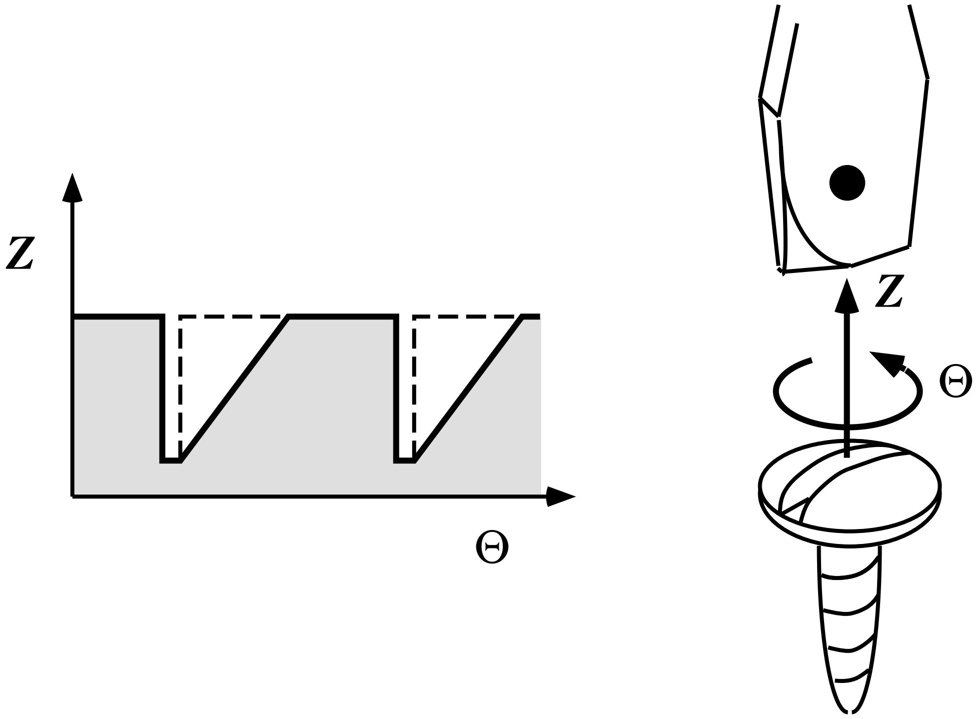


Figure 1.7: The "one way screwdriver", another possible instantiation of a modified set of motion constraints.

1.2.1 Key Issues

The hypothesis underlying the previous examples is that motion constraints can serve as a useful domain in which to represent, reason about, and design functional interactions between object shapes. This hypothesis encompasses two key issues:

Representation – How may function for different example domains be efficiently captured in terms of motion constraints?

Synthesis – How can the motion constraint representations be transformed into specifications of functional object shapes?

With the issue of representation we are concerned with the level of complexity in the construction of motion constraints as well as the accessibility of the representations to both humans and automated algorithms. With respect to synthesis, we are concerned essentially with the inverse of the process of generating motion constraints. Given two object shapes we may generate a precise description of their interaction in terms of motion constraints, but the inverse problem of generating shape is more problematic. First, we must have a precise *a priori* description of the motion constraints that we wish to impose, whereas often in design we may have only a partial or imprecise specification of the *class* of motions that we desire. Second, the process of inverting motion constraints is not uniquely defined mathematically. For the one way screw–driver examples in Figures 1.6 and 1.7, we essentially assumed that the desired motion constraint boundaries were known precisely and that either the screwdriver or screw head shapes were fixed. The resulting screw head or screwdriver contours were generated by sweeping the fixed shape along the prescribed constraints. We note that, although apparently successful in both of these examples, such a generation strategy is not in general guaranteed to produce the desired results (see Section 4.1.1).

1.2.2 Motivation

Shape interactions provide an important functional component of a number of systems. Numerous mechanical components, including simple mechanical pairs such as gears and cams, derive their function from shape interactions. Fasteners and drivers like those discussed earlier, as well as connectors for structural, electrical, pneumatic, and fluid applications are also dependent on shape interactions for function. In manufacturing, shape interactions have been identified as among the most important factors in mechanical assembly [16, 80], and are crucial in designing parts orienting systems that are typically among the most expensive components of many assembly systems [8].

Another reason for choosing motion constraints as a representation is the lack of suitable alternatives. As we illustrated in the earlier examples, function is derived explicitly from the shape interactions that constrain motion. Most existing design tools, such as Computer Aided Design (CAD) systems, focus on modeling the geometry of individual components in isolation. Individual shapes are considered explicitly, but any potential shape interactions are, for the most part, implicit in the geometrical representation. Consideration of shape interactions in such systems, if present at all, is usually limited to checking for interference between parts undergoing prescribed motions, typically along or around fixed axes. Interference checking, however, does not capture the potential functionality of object shapes for constraining motion. For some specialized applications, attempts have been made to divide individual object shapes into equivalence classes. As we saw in the part–feeder examples of Figure 1.3, however, similarity in shape does not necessarily correspond to similarity in function.

Shape design based on motion constraints is presently practiced for a limited set of well defined fixed-axis mechanisms such as cams and gears. In cam design, for example, the desired motion of a fixed shape cam follower is plotted as a function of cam rotation. The motion of the follower relative to the cam plate is then used to generate a tool path for cutting the corresponding cam shape. Similarly, gear profiles are often generated directly by hobbing or rolling processes where a fixed cutter shape is used to generate the complementary gear shape. Both of these processes are similar in concept to the one-way fastener synthesis example in Figure 1.6. However, few design tools utilizing motion constraints exist for other classes of shape interactions, and none are presently able to deal with non-fixed-axis devices. An extensive amount of work has been done in the area of analyzing the motions of mechanisms constructed from lower order kinematic pairs (i.e. revolute and prismatic joints). Techniques to specify the parameters for such mechanisms, such as link lengths for mechanisms of known topology, have also been developed and incorporated into computer aided design tools. However, these techniques are not suitable for more general kinematic interactions that cannot be characterized only in terms of interconnected links and joints.

1.2.3 Goals and Applications

The main goal of this research is to develop a representation and design *language* based on motion constraints that will allow us to reason about and create functional shape interactions. More specifically, we wish to:

1. Develop a precise and accessible representation for modeling and reasoning about functional shape in terms of motion constraints.

2. Develop the tools and methodology required to manipulate the motion constraint representations consistently in order to achieve desired functional behavior from shape interactions.

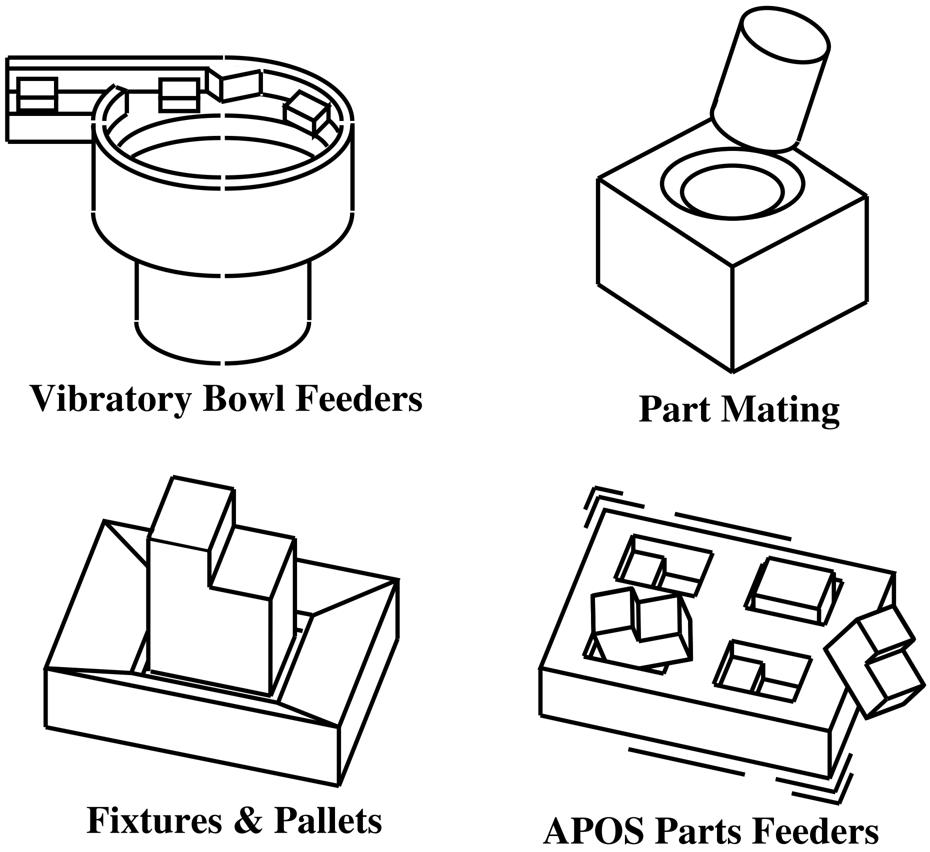


Figure 1.8: Four application domains.

These two sub-goals address directly the key issues of functional representation and synthesis identified earlier.

In addition to the above goals, we would like to be able to address some of the limitations of existing representations and techniques mentioned in Section 1.2.2. In particular, we would like to be able to represent and design more general shape interactions (i.e. other than pre-defined linkages and pins, revolute and prismatic joints, and fixed-axis mechanisms). We also would like to apply the representations and tools across a range of specific application domains in order to determine to what extent such representations are able to generalize functional characteristics beyond individual examples. Ideally these representations will enable us to identify or create functional shapes that otherwise might not have been generated.

We will judge the motion constraint representations and synthesis tools by the degree to which they enable us to perform analysis and design within a set of well defined example domains. To do this, we will specifically consider four application domains: compliant assembly, vibratory bowl feeders, assembly fixtures, and the APOS vibratory feeding system. Simplified examples of these four domains are shown in Figure 1.8. The representations and tools developed in this report will be used to analyze and reason about the functional characteristics of examples from each of the application domains, and to perform design in the first two. We will discuss these examples in greater detail in Chapter 3.

1.3 Background and Related Work

This research draws upon work in a number of different fields. Among the fields we consider to be most closely related are:

- Geometrical modeling and kinematics.
- Robot motion planning.
- Qualitative reasoning.
- Design methodology and computer aided engineering.
- Simulation and visualization of physical processes.

Kinematics and Mechanisms

In his seminal work on kinematics published in 1876, Reuleaux [66] introduced a model for mechanisms consisting of chains of kinematic pairs, which formed the lowest level of functional decomposition. He divided kinematic pairs into two classes: lower pairs and higher pairs. Lower pairs consist of objects in contact along a surface, and are comprised of six basic types: revolute, prismatic, helical, cylindrical, spherical and planar joints. Higher pairs involve objects in contact along a line or point, such as meshing gears, and are infinite in number. Reuleaux noted that all mechanisms can be derived from combinations of both lower and higher kinematic pairs. Although the number of basic mechanisms that may be composed are too numerous to mention, handbooks and catalogs containing some of the more commonly used and interesting mechanisms have been compiled. One example is an encyclopedia of mechanisms, together with textual descriptions of their function, compiled by Artobolevsky [2].

A large body of knowledge exists for analyzing the motions of mechanisms constructed from the six lower order kinematic pairs (see McCarthy [56]). A subset of the higher kinematic pairs, including many fixed axis mechanisms such as gear trains or cams & followers, have been analyzed extensively and special purpose synthesis techniques developed (see Paul [62] and Shigley [69] for examples). More recently, computationally based analysis techniques combined with synthesis tools to automatically select parameters for mechanisms of fixed topology, such as link lengths, have been developed by a number of researchers (see Bodduluri & McCarthy [6], Kramer [47], and Hoeltzel & Chieng [39]).

Shape Synthesis for Kinematic Pairs

The techniques and design tools described above are limited to the selection and manipulation of parameters for pre-determined kinematic pair types. Joskowicz & Addanki [43] outline an approach for modifying and generating the shape profiles of kinematic pairs from specifications of desired input-output motion relationships. Gupta & Jakiela [38] developed a shape design system that takes as input two planar shapes, one of which is fixed, and a functional diagram relating the relative motion of the two objects. As the fixed shape is swept along a prescribed motion path it is used to "carve" out the other object's shape, analogous to moving a hot knife through "computational butter".

Robot Motion Planning

Work in the field of robot motion planning has considered numerous aspects of the relationship between shape and motion. Tasks such as finding a collision free path for a robot among obstacles and the automatic synthesis of robot motions from high level task specifications have been extensively studied, and a number of powerful representations and analysis tools have been developed.

An important problem identified early on in the development of computer controlled manipulators was that of planning a collision free path of a manipulator among obstacles. Udupa [74] introduced a representation in which the problem of moving a robot among obstacles was transformed to an approximately equivalent and simpler problem of moving a point among transformed obstacles. Lozano-Pérez [49] formalized this idea by constructing the *configuration space*, whose axes are the robot's degrees of freedom, in which constraints on the robot's motion due to interactions with obstacles in the robot's environment could be represented directly as constraints on the motion of the robot. More generally, configuration space is the parameter space representing the relative locations (including position and orientation) of rigid objects. Although it has been used extensively in robotics and planning to model kinematic constraints imposed on the set of legal motions of objects by their shapes, the underlying idea of using a parameter space analysis has a long history in physics.

Construction of Configuration Space

Many algorithms for constructing motion constraint representations in configuration space have been developed for various motion planning and analysis applications, and the function and performance of these algorithms depends heavily on their intended application. The two primary applications for configuration space in robotics are the planning of collision free paths for manipulators and the analysis of the motion of objects in contact. Planning for collision avoidance typically involves searching among the free regions between obstacles in configuration space (see Lozano-Pérez [49]). The representations and algorithms constructed for this task therefore focus on characterizing the free and occluded *regions* of configuration space. Configuration space representations used for the analysis of motions of objects in contact focus less on the distinction between free and occluded space and more on the exact nature of the boundaries separating the two – the constraint *surfaces* determined by object interactions.

Lozano-Pérez [49] developed a simple and efficient algorithm for computing the boundary of the (x, y) configuration space obstacle formed by the interaction of two polygons without rotations. The algorithm is based on ordering the edges from the polygons by their orientation (measured counterclockwise), with the resulting list of ordered edges describing the (possibly non-simple) polygon forming the obstacle boundary. Avinaim et. al. [3] developed an exact representation of the full (x, y, θ) configuration space obstacle formed by two interacting polygons. Donald [22] developed and implemented an algorithm for planning collision free paths of three dimensional polyhedra with six degrees of freedom. In his representation, Donald described the five-dimensional constraint surfaces (submanifolds) in the complete six-dimensional configuration space. Bajaj & Kim [4] extended the class of modeled object shapes beyond polygons by developing algorithms to construct the (x, y) obstacle boundary for two interacting objects represented by segments of algebraic curves.

To improve the runtime performance of motion planning systems, a number of researchers have developed efficient algorithms to compute approximations for obstacles in configuration space. Lengyel et. al. [48] implemented a real-time motion planning system for polygonal objects by utilizing existing computer graphics hardware (depth buffer) to rasterize fixed-rotation slices of obstacles in configuration space. Branicky & Newman [11] developed and implemented algorithms to rapidly compute approximate configuration space obstacles for multi-link manipulators moving among polyhedral obstacles. Lozano-Pérez & O'Donnell [52] utilized a parallel architecture computer to rapidly compute and search for paths among obstacles in a six-dimensional configuration space generated for a six link manipulator. They achieved good performance by utilizing inherent symmetries in the structure of configuration space obstacles for revolute joint manipulators with intersecting axes, in order to represent and encode the obstacles within a recursive data structure that may be quickly computed and searched by massively parallel algorithms. A complete task level planning system named *Handey*, incorporating an implementation of the above algorithm in addition to a number of other algorithms addressing various aspects of robot motion planning, is described in Lozano-Pérez, et. al. [53].

Sensorless manipulation using task mechanics

Sensorless manipulation of objects is an important area of motion planning that often relies heavily on a detailed understanding of the underlying mechanics of a task. Unlike sensor based methods, such as robots coupled with vision systems, sensorless tasks are typically executed open loop, relying on the inherent task mechanics to reduce the effects of uncertainty.

A number of researchers have developed models of motion constraints imposed by the *mechanics* of object interactions that complement the kinematic constraints on motion described above in the various obstacle representations in configuration space. Mason [55] realized that motion constraint surfaces in configuration space could be extended to represent dynamic properties of manipulation. In particular, Mason observed that the surfaces of the configuration space obstacle possess many of the same physical properties attributed to 'real' surfaces, such as friction and the ability to generate reaction forces.

Erdmann [26] developed an extension of a common geometric representation of Coulomb friction – the friction cone – into an equivalent configuration space representation that includes reaction torques as well as forces. Goyal [40] examined the relationship between planar friction and sliding and developed the concept of motion limit surfaces that relate force-torque conditions to instantaneous motions for *known* sets of planar surface contacts. Peshkin [63] determined bounds on the instantaneous center of rotation for a planar object sliding on a surface where the distribution of contact forces was unknown. Wang [76] extended the class of modeled object interactions to include the mechanics of impacts among planar polygons.

Part Orienting

The task of part orienting is primarily concerned with the problem of reducing uncertainty and is conceptually very similar to the problems of both sensor-based and sensorless manipulation. Like manipulation strategies, we may divide feeders into two broad classes: sensor-based and sensorless.

Sensorless Orienting

Vibratory bowl feeders are among the most common type of sensorless feeder in widespread use. The seminal work by Boothroyd et. al. [8] presents a comprehensive and in depth analysis of feeding and orienting techniques in general, and in particular bowl feeders. They conducted numerous experiments to determine the probability distribution for the resting aspects of parts and developed techniques to compute the throughput efficiency of vibratory bowl feeder systems. The main result of this work is a handbook of feeders indexed by a taxonomy of basic part geometries. This handbook serves as an essential tool for designers and manufacturing engineers in making an *initial* selection of feeder types and geometries appropriate for a given set of parts.

The **(APOS)** (**A**dvanced **P**arts **O**rienting **S**ystem) developed by Sony and described by Shirai & Saito [70] represents a simple and efficient hardware implementation of vibratory feeding that combines reusable system hardware with integrated palletizing and part transport. **APOS** has been successfully applied in a wide variety of manufacturing systems both inside and outside Sony. The primary challenge in using the system is the initial design of pallets that capture, sort, and hold parts for assembly – hence its interest to us in the context of shape design from motion constraints. Moncevicz [58] describes an interesting approach that seeks to further integrate orienting and assembly operations. It is based on a modification to the basic APOS system in which component parts are both oriented *and* assembled in vibratory pallets where the pallets are themselves subassemblies, an approach they refer to as "shake'n make" assembly. In another interesting approach, Singer & Seering [71] distinguish and separate part orientations using differences in the dynamic properties of parts by running the parts over a small fence placed across a moving track.

Part Orienting in the Context of Planning

Natarajan [59] considered a number of general theoretical and computational aspects of part orienting as a sensorless manipulation task. Erdmann & Mason [27] developed and implemented an algorithm to generate sequences of sensorless tray tilting motions designed to place a randomly oriented part into one corner of the tray in a known orientation. Goldberg [34] developed algorithms to generate sequences of planar grasps using a frictionless parallel jaw gripper to orient a polygonal part of known shape in the plane. Although a robot was used to perform the grasping motions, no sensing was done.

Sensor-Based Orienting

Numerous systems consisting of combinations of robots, cameras, lasers, photodiodes, contact switches, actuators, etc. have been developed and implemented. Gordon [35] provides both a good example of an integrated closed loop orienting and assembly system using a laser, vision system and a robot, as well as an excellent survey of related sensing and orienting techniques.

Qualitative Reasoning

Work in the field of automated qualitative reasoning has examined the relationship between the shape of kinematic pairs and their corresponding function with the goal of extracting abstract descriptions of function from form. Faltings [29] introduced the *place vocabulary*, an abstraction for mechanism function derived from the topology of free regions in the configuration spaces of the kinematic pairs composing the mechanism. A place vocabulary consists of a graph representation in which bounded free regions in configuration space are represented as nodes and connections between the regions as arcs. The resulting graph structure embeds a compact encoding for the topology of free regions in configuration space that may be parsed to obtain the functional attributes of the underlying mechanism.

Joskowicz [44] also uses boundaries and regions in the configuration space of a mechanism to perform qualitative analysis of mechanism behavior. In addition, he introduced a heuristic algorithm for designing the shapes of mechanism components from descriptions of desired behavior represented either in terms of configuration space maps or functional relationships between the input and output parameters defining the mechanism's configuration space. Joskowicz & Sachs [45] extended and implemented this work on kinematic constraints to include the dynamic behavior of mechanisms. One result is a system for the automated modeling and analysis of planar mechanisms consisting of chained kinematic pairs. The system first constructs the 2D configuration space for each kinematic pair in the mechanism and then automatically explores both the dynamic and kinematic behavior in each of the coupled configuration space regions.

Bourne et. al. [9] used configuration space as a domain for examining the relationship between machining tolerances for parts and the functionality of those parts in a mechanism. Specifically, they searched for parametric variations that changed the topology of free space regions in a mechanism's configuration space in order to both highlight sensitive design parameters and to derive tolerance constraints that were related directly to the intended function of a given mechanism.

Design Methodology

Nevins & Whitney [60] present an overview and series of detailed case studies on the development and implementation of *concurrent design* strategies for products and processes. The broad aim of concurrent engineering is to consider multiple aspects of and constraints on a product's function, manufacture, use and (recently) disposal as early on in the design process as possible.

One area of concurrent design dealing with product assembly is design for assembly (DFA), extensively developed by Boothroyd and others [7]. DFA methodologies consist of case studies, design rules and heuristics aimed at reducing the overall number of component parts and fasteners in a product's design, as well as making the parts easier to identify and orient. Jakiela [42] implemented a design environment that utilized encoded design for assembly (DFA) rules to make suggestions for changes to the input geometry during the design session. Whitney et. al. [80] presented detailed models, analyses and experiments on the performance of various chamfer profiles during the one-point contact phase of peg-in-hole assembly. Caine [16] examined the effect of a number of chamfer profiles on the jamming and wedging characteristics of planar peg-in-hole assembly during two-point contact. De Fazio et. al. [21] have implemented a feature based interactive CAD environment for analyzing the assemblability of parts using a graph of liaison diagrams encoding the desired relationships between collections of parts in order to select the proper assembly sequence.

Simulation and Visualization

The primary roles of simulation in design is that of model verification and troubleshooting via exploration of the functional properties of the system under consideration. One of the difficulties frequently encountered in simulation involves inherently discontinuous phenomenon, such as impacts, that result in constantly changing boundary conditions that require frequent changes to the system models.

Gilmore & Streit [33] developed a system for predicting motion under multiple discontinuous contacts using a rule based algorithm that determines changes to constraints and automatically reformulates the dynamical equations accordingly. The system was applied to the analysis of a parts feeder for planar parts consisting of a sequence of angled fences. Donald & Pai [24] utilized a simplified configuration space representation to analyze and simulate the motion of rigid planar parts with compliantly connected "snap" features moving in a plane. The resulting system was used to help redesign the shape of the interacting planar parts for more reliable assembly.

Simulation techniques have also been applied in interactive graphical environments. Witkin [77] introduced a reformulation of dynamical and constraint equations describing a system so that constraints could be rapidly added and removed as the equations were integrated numerically. In one application of this technique, graphical entities could be created, linked and unlinked interactively in real time by a user. Related techniques developed in the rapidly evolving visualization field have found application in such diverse fields as computational fluid mechanics, meteorology, resource extraction, and computational biology (see Patrikalakis [61] for numerous examples). In all cases, the basic goal of such technology is to represent complex or large sets of data within a unified representation that aids in reasoning about and manipulating the data.

Particularly Relevant Work

In his Ph.D. thesis, Brost [13] developed and implemented an exact representation for the (x, y, θ) configuration space obstacle formed by two interacting polygons. Unlike collision-free planning applications, Brost's implementation focuses on constructing a detailed motion constraint representation, including the mechanics of object contact, that is suitable for the detailed analysis and planning of object interactions. In addition to kinematic constraints between planar objects, Brost developed a series of algorithms for computing regions of possible static equilibrium and bounded regions defining configurations reachable from initial positions in the presence of positional and control uncertainty. Brost applied these algorithms to the tasks of analyzing and planning robot pushing motions and dropping of parts into orienting fixtures.

The representations and implementation developed by Brost preceded those presented in this report, and there a number of similarities and differences between the two. Similarities between the two implementations include:

- Both implementations consider interactions among objects modeled as polygons moving in the plane with three degrees of freedom.[3]
- Both implementations compute an exact representation of the kinematic motion constraint *surfaces*, represented in (x, y, θ) configuration space, produced by planar polygon interactions.
- Both implementations model the mechanics of object interactions, including coulomb friction.

The primary objective of Brost's implementation is the automatic construction of plans, consisting of either pushing or dropping motions, represented geometrically as regions in configuration space backprojected from specified goal states. This is in contrast to the forward projections from specified starting positions/regions that are computed by cspace-shell for the purpose of visualization and analysis by a user. Some of the specific differences between Brost's implementation and the cspace-shell implementation presented in this report include:

- Brost treats the shapes of objects as static variables since he is concerned with planning motions for objects of known shape. In cspace-shell, however, it is the modification of object shapes that is of primary concern. As a result, Brost's implementation precomputes the full topology of the configuration space obstacle whereas, for reasons of computational speed, cspace-shell computes and displays the complete set of individual contact facets, but only computes the

[3] Brost's algorithms implicitly consider both polygons to be fully supported by an underlying planar surface. Hence there is no explicit consideration of limited support due to interactions with a non-infinite supporting plane profile, as with the track in the bowl feeder examples in this report.

local topology (facet adjacencies and intersections) that affects the integrated motion paths.

- Brost explicitly considers and models uncertainty in his representations and algorithms, both symbolically and numerically, in order to compute motion plans that are robust. The present implementation of cspace-shell performs only exact numerical computations.

The following is a comparison between specific components Brost's implementation (highlighted) and cspace-shell:

(CO) produces an exact metric and topological description of the (x, y, θ) kinematic motion constraints for two input polygons. It is comparable to the facet generation and (local) topological checking performed during motion integration in cspace-shell. However, as mentioned above, cspace-shell does not precompute the full topology of the configuration space obstacle, which Brost's implementation must do in order to perform backprojection computations.

$(STATIC)$ computes and labels regions on the surface of the motion constraint set that *may* correspond to static equilibrium under specified applied forces and uncertainty. There is nothing directly comparable in cspace-shell, although static equilibrium (without uncertainty) is checked during motion integration in order to detect motion termination.

(BP_e) produces energy "puddles" on the surface of the motion constraint set that define the set of initial positions (and orientations) from which an object may be dropped in a gravity field and still be guaranteed to come to rest in a specified location. These energy puddles are equivalent to the conservative energy bounded forward projections *described* in Section 2.4.2 with $e = 1$. Two important differences are that BP_e is at least partially implemented and that the resulting regions are backprojected, i.e. are generated backwards from desired goal states. The energy bounded forward projections in this report have not been implemented.

(BP_i) computes backprojected regions by expanding the set of points on the surface of the configuration space obstacle from which a desired goal will be reached. The resulting surface regions are then *lifted* from the surface in order to form volumes in configuration space that define the set of initial positions (and orientations) from which an object may be reliably pushed into a specified location in the presence of uncertainty. BP_i is similar to, but significantly more general than, the numerically integrated forward projections computed by cspace-shell. In addition, BP_i can model higher order dynamics whereas the present implementation of cspace-shell assumes only quasi-static motions.

Another piece of work particularly relevant to this research was described in an unpublished research memo by Lozano-Pérez [51], who proposed both a representation of function from shape in terms of motion constraints, and the view of the process of shape design as an inverse of the motion planning problem. He also suggested vibratory bowl feeder design as a promising domain in which to develop and test these ideas.

Specifically, Lozano-Pérez proposed a design model in which feeder motions were characterized and classified into a basic set of primitive motions. These primitive motions would serve as a vocabulary with which the desired motions of parts to be oriented could be concisely represented. To generate feeder geometries, characteristic motion paths would be composed from this vocabulary and parts would be swept along those paths, effectively cutting out the desired feeder. As we will see in Chapter 4, the geometries generated by such swept motions provide the necessary but not the sufficient conditions to guarantee that the desired motions will be achieved. Although these ideas were not developed further or implemented by Lozano-Pérez, they provided the inspiration and underlying conceptual framework that guided the bulk of the work described in this report.

1.4 Contributions of the Research

The contributions of this research lie both in the underlying concept of using motion constraints as a paradigm for shape design, and in the representations and tools developed for this purpose. The major contributions of this research include:

- **Motion Constraint Based Techniques for the Design of Functional Shapes.** This research has demonstrated that motion constraints may be used as the basis for *design* of functional shapes as well as for the analysis of functional shapes.

- **Functional Constraint Representations.** We have developed mathematically precise and computationally accessible functional representations in terms of motion constraints for the four application domains shown in Figure 1.8: compliant assembly of rigid parts, vibratory bowl feeders, part fixtures, and APOS vibratory parts feeders.

- **Design Tools and Methodologies for Generating Functional Shape Interactions.** We developed and implemented a set of tools and methodologies applicable to the design of vibratory bowl feeders and compliant assemblies.

- **Computation of Planar Support.** We developed an efficient algorithm for computing the condition of support for an object by a planar support surface under the effects of gravity.

- **Lower Dimensional Mappings of Transitions to Higher Dimensional Motions.** We developed a representation that maps the transition from lower d.o.f. constrained motions to higher d.o.f. motions as a result of changes in planar support. This mapping allowed the essential characteristics of complex object motion to be captured within a simpler representation.

- **Interactive Computational Environment for Functional Shape Design.** We developed and implemented an interactive computational environment for shape design based on a "near" real-time motion constraint generation, visualization and manipulation tool. This system also demonstrated that the computation of constraints in configuration space, for objects with three degrees of freedom, may be computed **quickly** for planar objects of "moderate" complexity.

- **Functional Shape Generation by Means of Swept Motions Doesn't Work.** We illustrated that shape generation techniques based on swept motion of fixed shape objects are not guaranteed to provide the intended motion constraints, illustrating the need for a more accurate and complete representation for analyzing and synthesizing shape interactions.

- **Dynamic Visualization of Coupling Between Motion Constraints and Shape Parameters.** We utilized the above computational environment to visualize, identify, and interactively explore the dynamic nature of constraint coupling. We introduced the notion of dynamic constraint visualization as a means of examining the neighborhood of a point in design space and highlighting the inherent limitations and constraints on achieving a desired set of functional properties.

1.5 Outline of the Report

Chapter 2 develops the detailed motion constraint representations for the class of objects that may be modeled as planar polygons with a maximum of three degrees of freedom (two translational, one rotational). The notion of motion constraints is extended to include both kinematic and non-kinematic constraints, and various types of motion forward projection are developed for various mechanics models.

Chapter 3 develops the four application domains introduced in Figure 1.8: compliant assembly, vibratory bowl feeders, assembly fixtures, and the APOS vibratory feeding system. The motion constraint representations developed in Chapter 2 are displayed in visual form and used to analyze and reason about the functional characteristics of examples from each of the application domains.

Chapter 4 extends the utility of the representations developed in Chapter 2 by introducing a series of tools to manipulate the motion constraints directly together with their underlying shapes. These tools are applied to the design of a set of functional shapes from two of the example domains introduced in Chapter 2: compliant assembly and vibratory bowl feeders. Methodologies are developed to address the inherent coupling and complexity of design in the two application domains.

Chapter 5 describes the implementation of the motion constraint representations and design tools on a graphics workstation. The major components of the implementation are outlined and discussed, and a number of optimizations required to generate and interactively manipulate motion constraints in near real-time are highlighted.

Chapter 6 summarizes the major concepts of the research and a discussion of the current limitations and possible future extensions of the approach.

Appendix A contains derivations of the models used to compute forward projection bounds for an object dropped from rest in a gravity field, and a conservative estimate of the maximum vertical height that may be reached by an object bouncing in contact with a vibrating table. Appendix B contains derivations of the curvature of contact facets along a curve on the surface of the facet, which are used to ensure accuracy bounds on the numerical path integration. Appendix C presents the primary data structures used in the implementation of the motion constraint analysis and design system.

Representing Function

Chapter 2

In this chapter we will develop the representations necessary to capture function in terms of motion constraints. These representations will provide the foundation upon which we will evaluate and manipulate both shape and other parameters as necessary to achieve desired functional characteristics.

2.1 Functional Motion Constraints

Precise Representations of Motion Constraints

We often use terms such as *guide*, *support* and *restrain* in describing the functions performed by interactions between shapes. These terms are used to refer to constraints on specific subclasses of motions that are qualitatively distinct with respect to the intended function of the constraint. Unfortunately, the meaning of the term *constrain* in the context of one example may be different in another, or equivalent to the meaning of *support* in yet another. Clearly the semantics of the above words are too vague and imprecise representation to be suitable for accurately characterizing function. What we need is a more precise representation; a model for function represented in terms of motion constraints.

Consider again the fastener and driver examples in Section 1.2. In describing the function embodied in the fastener-driver interactions in Figures 1.4 and 1.5, we adopted a representation based on the space of parameters describing relative object positions. The shaded regions illustrated in these figures represented driver positions that were unreachable due to the presence of the screw head, and the boundaries between shaded (occluded) and unshaded (free) regions represented kinematic constraints on the object motions due to contact interactions between their shapes. Looking at these boundaries another way, we may view them as constraining where one object *can* and *cannot* go relative to another.

Kinematic and Non-Kinematic Motion Constraints

Is the above representation for kinematic motion constraints sufficient to capture function? Consider the example of a cup sitting on a table. We can say that the interaction between the shape of the cup and the table constrains the cup to remain *on or above* the table's surface. Expressed in a space representing the position and orientation of the cup relative to the table, similar to that used for the fastener-driver examples, the point representing the configuration of the cup may placed anywhere in the free region bounded by the constraint surfaces formed by the interaction between the cup and table. But what if we now wish to answer the question "does the table *support* the cup?" Clearly this representation alone is not sufficient to determine the behavior of such a system. What's missing is an additional set of **non-kinematic** motion constraints that, when combined with the kinematic motion constraints, will tell us not only where an object can and cannot go, but where it *will* go under given conditions.

2.2 A Parameter Space for Motions

We represent the position of one object relative to another object as a transformation between coordinate frames attached to those objects. The parameters used to define this transformation are the parameters that will be used to express a relative motion between the objects. These *configuration* parameters, which may include translations as well as rotations, are used to define a space in which object positions are given as points and object motions by trajectories – a configuration space. We may consider the configuration space a form of kinematic state space for representing motions that is a direct analogy to the generalized phase space used to describe the behavior of a second order dynamical system – without the velocity information. For many applications we may furthermore assume one of a pair of interacting objects to be fixed in a global reference frame so that the relative position of the moving object becomes a specification of absolute position.

Configuration space is a natural choice for our application as it allows us to represent both motions and constraints on motions with the same set of variables. As we shall see, shapes will be combined via contact interactions to form an explicit representation of motion constraint, while the shapes themselves remain implicit within the resulting representation. The dimensionality of the configuration space will depend on the number of parameters necessary to describe any possible object motion; the maximum number of degrees of freedom of the system under consideration. The number of degrees of freedom for general motions of individual three dimensional objects, and hence the number of dimensions of the configuration space, will be six: three translational parameters and three rotational parameters.

Reducing the degrees of freedom for an object is desirable both in terms of the number of parameters that must be considered and the complexity of the analysis that must be performed. From the standpoint of human visualization of function, three dimensions is the realistic limit within which our spatial reasoning abilities will be useful in understanding motions and constraints represented in configuration space. Furthermore, the complexity of computing constraints in configuration space grows *polynomially* with the geometric size of the objects, and *exponentially* with the degrees of freedom (see Canny [18]).

Fortunately, it will often be sufficient to consider only a subset of the possible motions of an object for a particular task. In some cases, symmetry will allow us to remove degrees of freedom that might be considered redundant; axisymmetric objects such as a cylindrical peg in hole modeled as planar objects is an example of one such case. In other cases the motions of objects may initially be constrained enough to require further consideration of only the remaining degrees of freedom, such as an object dropped onto a table that then slides across the table surface (without tipping). And in still other cases, object motions may be constrained by design from the outset, such as gears and cams rotating about fixed axes or pistons sliding within cylinders. Finally, for those domains where more than three degrees of freedom must be considered, it is often possible to subdivide or otherwise isolate different aspects of function that individually may constitute fewer degrees of freedom. For example, by considering the motion of an object sliding in the plane that may tip as a series of sub-motions consisting of purely planar motions connected by tipping motions, we tradeoff simpler models in return for a greater number of those models that must be generated. We will consider a number of such simplifications later on in this chapter.

For most of this report we will consider in detail objects constrained to have three or fewer degrees of freedom. Specifically, we will model and analyze in detail objects that move only within a plane. Motions will consist of displacements in x and y of a reference point attached to the moving object, and a rotation in θ of the object about a normal to the plane of motion. We will occasionally refer to the three dimensional configuration space described by these parameters as **2+1D** ($\mathcal{R}^2 + SO(1)$) in order to distinguish it from **3D** ($\mathcal{R}^3$).

2.3 Kinematic Constraints

Since the position of an object is represented as a point in configuration space, constraints on an object's motion due to shape-shape interactions must be transformed so as to be local to a point – shape-shape interactions map to point-surface interactions in configuration space. The characteristics of these constraint surfaces are the subject of this section, although we will defer a detailed treatment of the techniques used to construct these surfaces until Section 5.3.1. We will begin by examining constraints for contacts between individual object feature pairs, and then address combinations of multiple contacts and their relationships to one another. In the discussion that follows we will assume that both the moving and stationary objects may be modeled as planar polygons.[1]

2.3.1 Individual Feature Contacts

We will consider explicitly only the vertex-edge contact configuration in modeling the interactions between individual features of two planar polygons. The two other possible feature pair contact configurations for planar polygons (vertex-vertex and edge-edge) will be treated as boundary cases to be represented among the set of multiple-feature contact interactions discussed in the next section. With the moving polygon labeled as **Polygon A** and the stationary polygon as **polygon B**, we refer the two contact types as **type A** (an edge of Polygon A touching a vertex of Polygon B) and **type B** (an edge of Polygon B touching a vertex of Polygon A) (see Lozano-Pérez [49]). Figure 2.1 illustrates these two cases, along with their corresponding motion constraint surfaces in configuration space, which we call contact *facets*. Each facet represents the complete set of positions in (x, y, θ) of the reference point of the moving polygon for which the corresponding vertex and edge features will remain in contact. Figure 2.1 illustrates both facet types along with their corresponding polygon contacts.

Mathematically, the contact facets are ruled surfaces generated by sweeping a bounded line segment corresponding to a polygon edge through the (x, y, θ) configuration space (see Section 5.3.1). For type A facets, an edge of the moving polygon can slide and change orientation while in contact with a vertex of the stationary polygon, resulting in a helical surface as shown in Figure 2.1. For type B facets, a vertex of the moving polygon can slide in contact with an edge of the stationary polygon that maintains a fixed orientation, resulting in a sinusoidal surface as shown. The boundaries of the facets represent the limits of motion in which the two features may remain in contact.

Intuitively, the constraint facet surfaces behave in the same way as real surfaces: forces applied to the surface at a point generate opposing reaction forces, sliding motions along the surface can generate frictional forces, and motions may break contact with but not penetrate the surface. Unlike conventional, or "real" surfaces, motions in contact with constraint facets explicitly combine components of translation and rotation. As a result, a facet's curvature in the θ direction reflects the *arc* through which the reference point of the moving object will move during a rotation. For a type B facet, a large degree of facet curvature results from a large distance from the reference point to the contacting vertex, little or no facet curvature corresponds to

[1] We note that although we consider only planar motions, the objects themselves may be fully three dimensional. In later sections we will discuss modifications to our models that address higher dimensional models of objects and motions.

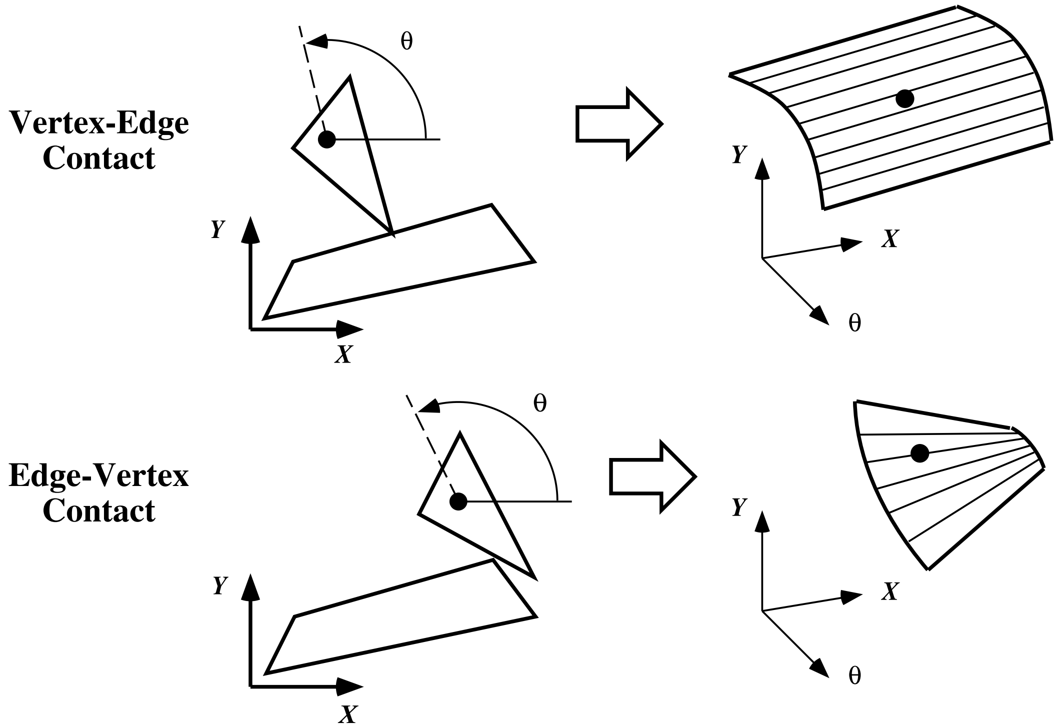


Figure 2.1: Type A (vertex-edge) and type B (edge-vertex) contacts between planar polygons.

a reference point very near the contacting vertex. A negative, or concave, curvature corresponds to a negative distance from the reference point to the contacting vertex (see Figure 2.2). The same arguments hold for Type A facets, although their curvature is somewhat more complex since the distance from the point of contact to the moving polygon's reference point varies with translational motion. Generally speaking, the greater the distance from the contact point to the reference point, the more curved the facet.

The facets and corresponding contact configurations shown in Figure 2.2 imply a certain sense of motion stability or instability with respect to their curvature. An object released in either configuration **(a)** or **(b)** would be unstable if we imagined gravity acting toward the bottom of the figure. By analogy, a point or small ball placed on either of the facets in **(a)** or **(b)** would tend to slide or roll off of the facet. Figure 2.2 **(c)**, on the other hand, intuitively seems to be a more stable configuration both in maintaining the position of the object shown as well as keeping the equivalent point or ball at the bottom of the "trough" in the concave facet. We will explore such interpretations of constraint facet shape, as well as the effects of various dynamic models on motions in contact with the facets, in later sections. Our purpose here is to try to convey an intuitive sense for the structure and interpretation of these constraints.

2.3.2 Contact Supersets

Type A and type B constraint facets allow us to represent all possible *individual* feature contacts between two polygons in configuration space. Constrained motions typically include *combinations* of and *transitions* between individual contacts. We therefore need to be able to represent the relationships among collections of feature contacts as well as individual contact constraints. Figure 2.3 illustrates a number of adjacent constraint facets representing contacts among consecutive polygon features. The boundaries between the facets mark transition contacts, in this case either vertex-vertex or edge-edge contacts, that can themselves be viewed as distinct contact conditions.

Figure 2.3 represents a subset of the larger union containing *all* constraint facets for two interacting polygons. This union, which we will refer to as the **Contact Superset**, or **CS**, forms a closed surface partitioning the configuration space into reachable and unreachable regions. Figure 2.4 shows the complete CS generated for two polygons. In this figure we can see many adjacencies between facets that resemble the subset shown in Figure 2.3. The CS for two convex polygons consists entirely of adjacent facets since all contact transitions are between consecutive edge and vertex features of the two interacting polygons. For polygons that are not strictly convex, however, it is possible to have contact transitions between polygon features that are not consecutive on the polygon's boundaries. Specifically, for non-convex polygons

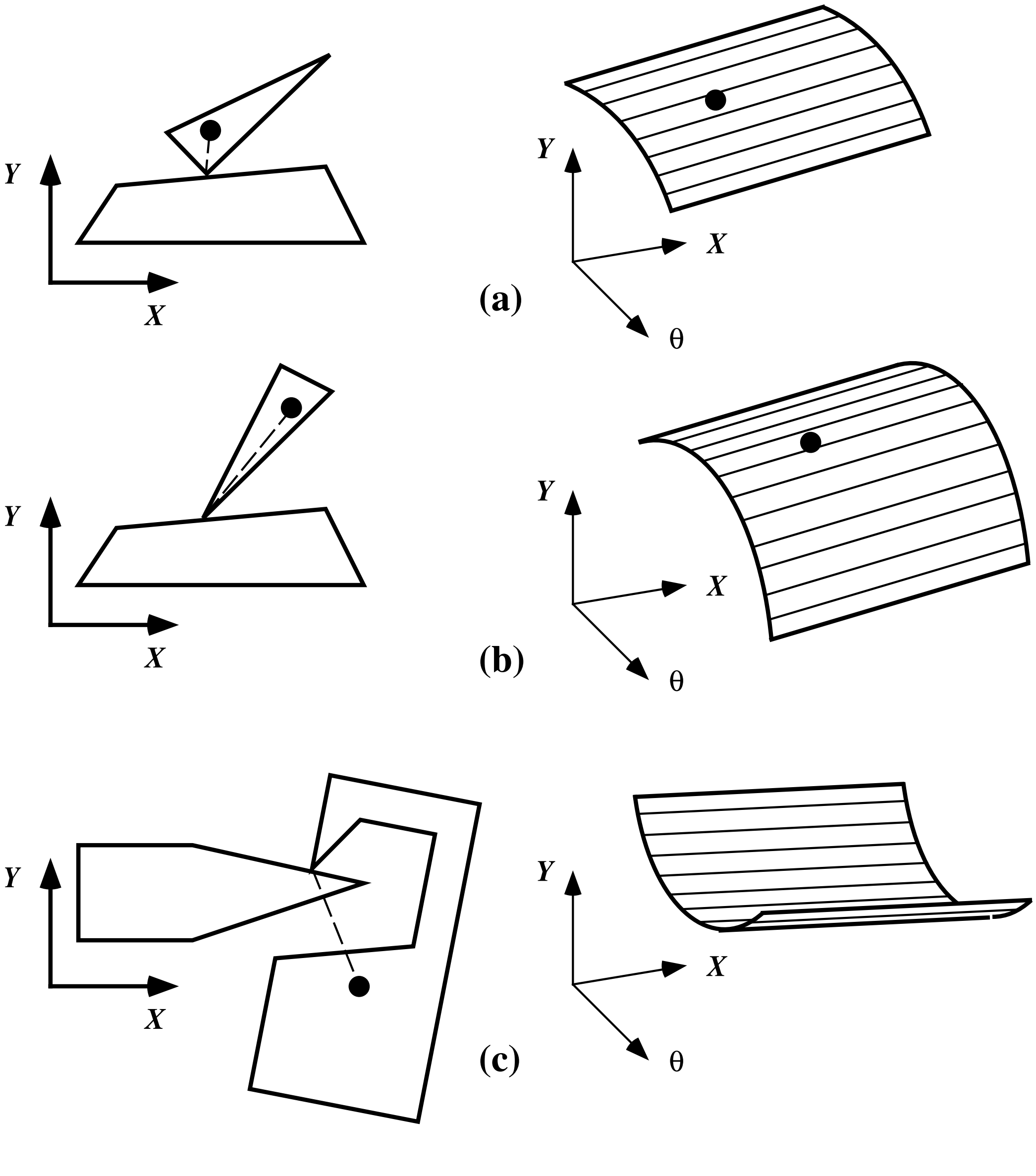


Figure 2.2: Type B constraint facets with different degrees of curvature.

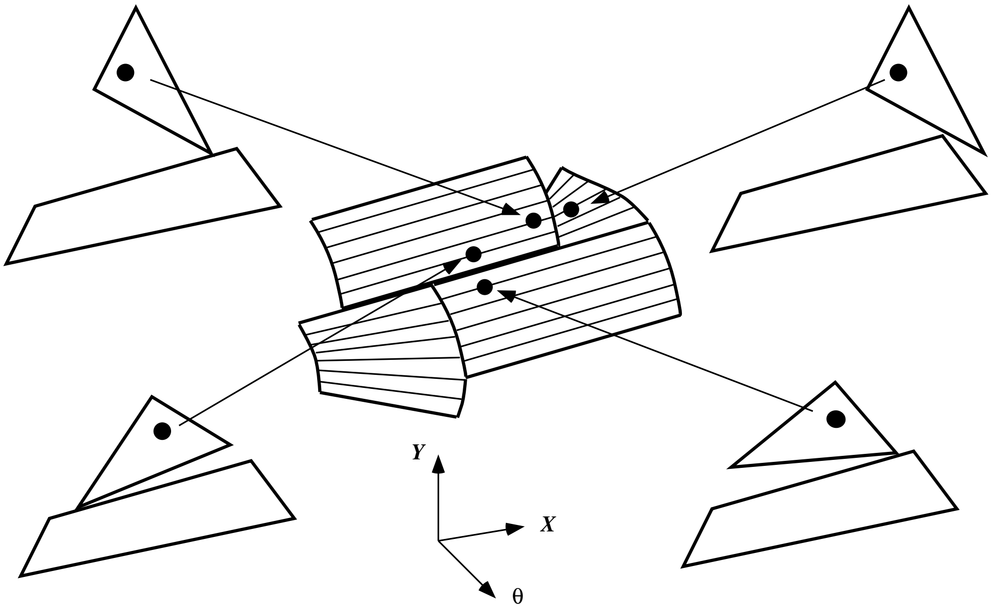


Figure 2.3: A collection of adjacent constraint facets.

we will find that some constraint facets may be partially or completely occluded by other facets. We recall that each constraint facet represents only the local motion constraints imposed by two interacting polygon features. In some cases, locally consistent motion constraints may be globally unreachable, as shown in Figure 2.5.

Topologically, the CS surface consists of facets, edges, and vertices.

- **Facet Contact:** Each facet corresponds to a single contact between one feature on each of the two interacting polygons. As mentioned earlier, a facet's shape, and in particular its curvature, is determined by the type of contact and the distance from the point of contact to the reference point of the moving object, the positions of which the facet surface represents. A point in contact with a constraint facet has two degrees of freedom.

- **Edge Contact:** Each edge is derived from either an adjacency or an intersection between two facets and corresponds to a contact between two pairs of object features.[2] Straight edges perpendicular to the θ dimension of configuration space arise from edge-edge contacts between polygons. They typically mark adjacencies between facets, and are usually concave (i.e. form "valleys" on the CS). A curved edge may arise from a vertex-vertex contact, or from an

[2]One of the object features in each of the two feature pairs could be the same feature, i.e. the same edge of one polygon might be in contact with two vertices of the other polygon.

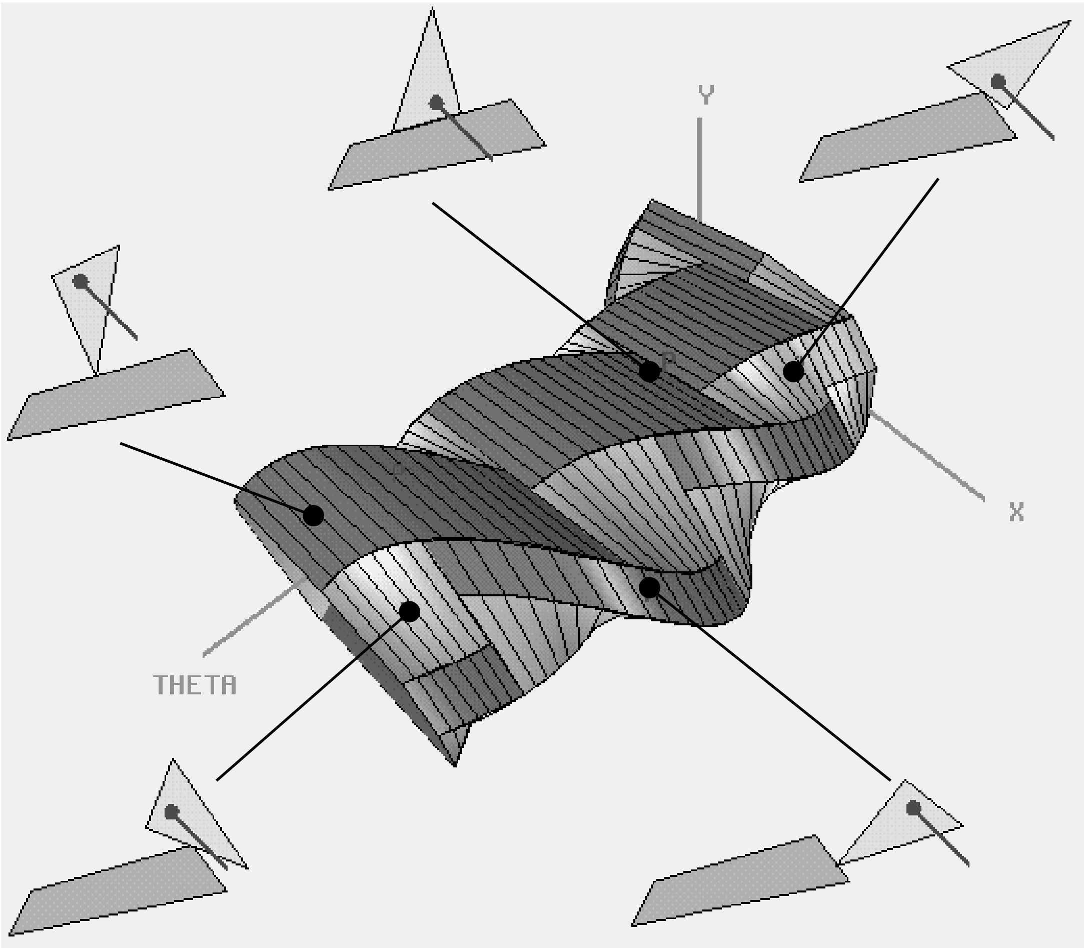


Figure 2.4: A contact superset for two planar polygons in (x, y, θ) configuration space.

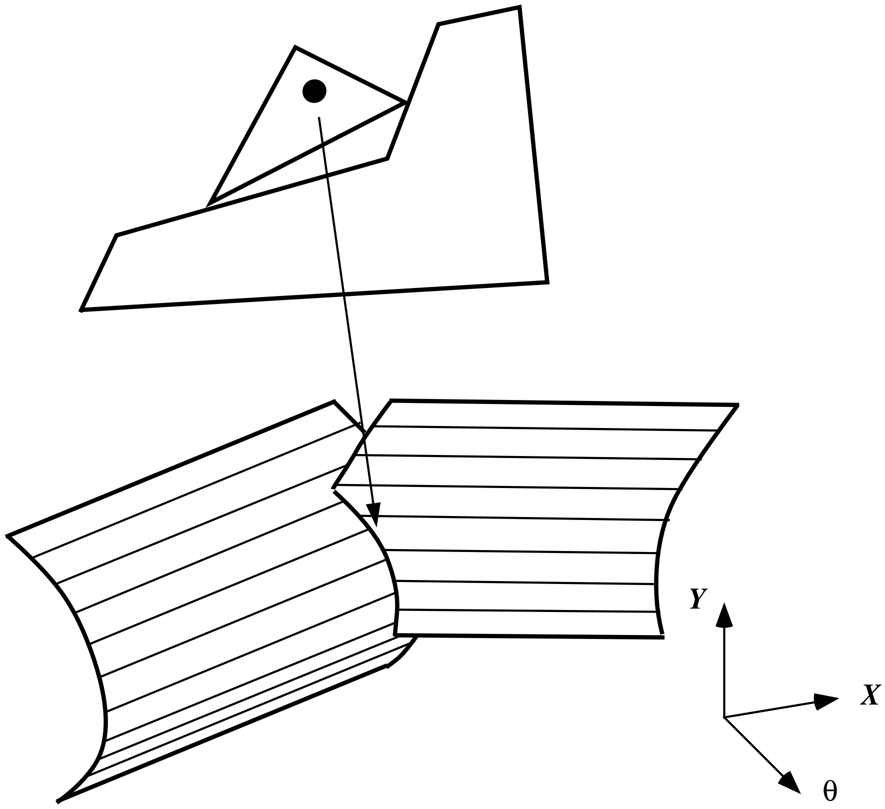


Figure 2.5: Intersecting facets and their corresponding polygon feature contacts.

intersection between two facets, which is usually concave. A point in contact with a CS edge has one degree of freedom.

- **Vertex Contact:** Each vertex is derived from three edges on the CS meeting at a point. Vertices may mark the point of adjacency between three facets, in which case the vertex is flat (i.e. neither concave nor convex). Vertices may also mark the point at which a (convex) adjacency edge between two facets intersects a third facet, producing a vertex between the convex edge and two concave edges. Finally, a vertex may mark the point of intersection between three facets, in which case the vertex is strictly concave. A point in contact with a vertex of the CS has zero degrees of freedom.

Figure 2.4 illustrates a number of the features described above, with the corresponding polygon contact configurations highlighted.

A great deal more could be, and in fact has been, written about the characteristics of facet, edge, and vertex features on the CS. An excellent treatment of the detailed characteristics and mathematical properties of constraint facets, edges, and vertices in the (x, y, θ) configuration space is given by Brost [13]. Again, here we are interested more in an intuitive understanding of these features and their significance in terms of motion constraints. Details on how we (implicitly) construct such features may be found in Section 5.4.1.

As we noted earlier, the configuration space representation transforms all motion constraints imposed by shape-shape interactions into point-surface interactions. Since all constraints in configuration space are local to a point, the *proximity* of constraints takes on a new and important meaning that is not present in "real" space representations.[3] As we saw in Figure 2.5, features on an object that are distant in terms of the object's geometry can give rise to proximal motion constraints that are made explicit on the CS surface. When we consider motions in contact with the CS surface later in this chapter, the proximity of constraint features on the CS surface will be an important factor in determining what motions can and will occur under given conditions.

2.4 Non-Kinematic Constraints

The set of kinematic constraints represented by the CS serves to partition the configuration space into reachable and unreachable configurations of the moving part relative to the stationary part. As noted in Section 2.1, the CS tells us where the moving object *can and cannot go* due to the presence of stationary object. We may further partition the reachable region(s) of configuration space by introducing other,

[3] We define proximity to be a measure of the magnitude of a motion in (x, y, θ) necessary to reach a specified contact state.

non-kinematic, constraints that are determined by factors other than shape. These sub-regions further constrain the set of possible positions/motions of the moving object and may range anywhere from large bounded regions down to one-dimensional space curves through the configuration space. Again, these additional constraints, together with the kinematic constraints in the CS, serve to give us a better idea of where the moving object *will go* for a given situation.

2.4.1 Contact Mechanics

Contact between objects represented by the CS surface may produce contact forces. These forces, which in turn affect the resulting object motions, are determined by both the geometry and material properties of the objects in contact.

Forces, Torques and Friction

Motion constraint surfaces in configuration space produce reaction forces in much the same way as real surfaces. A reaction force produced by the CS surface will have components along the surface normal and, if there is non-zero friction, tangent to the surface as well. One key difference between the forces in real space and forces in (x, y, θ) configuration space is that the θ component actually corresponds to a scaled torque.

Friction, which depends on the material type and surface properties of the contacting objects, determines the range of contact forces (and torques) that may be generated by a contact. Figure 2.6 (a) illustrates the Coulomb model of friction expressed geometrically as the *friction cone* for a single contact. The friction cone spans the range of reaction forces that may be generated in response to an applied force at the point of contact. The half-angle ϕ of the friction cone is given by:

$$\phi = arctan\left(\frac{1}{\mu}\right)$$

where μ is the coefficient of friction. External forces whose direction lies inside the angle range spanned by the friction cone are exactly canceled by the corresponding reaction force, whereas external forces oriented outside the cone will be only partially canceled by a reaction force lying on one of the bounding rays of the cone. In either case, a net torque will be generated about the point of contact for any external force whose line of application does not pass through the point of contact. A configuration space analog of the cartesian space friction cone that captures the torques associated with the reaction force is shown in Figure 2.6 **(b)** (see Erdmann [26]). The range of reaction torques associated with the reaction forces spanned by the cartesian space friction cone acts to tilt and rotate the configuration space friction cone with respect to the contact facet normal in (x, y, θ). Actually, the cartesian space friction

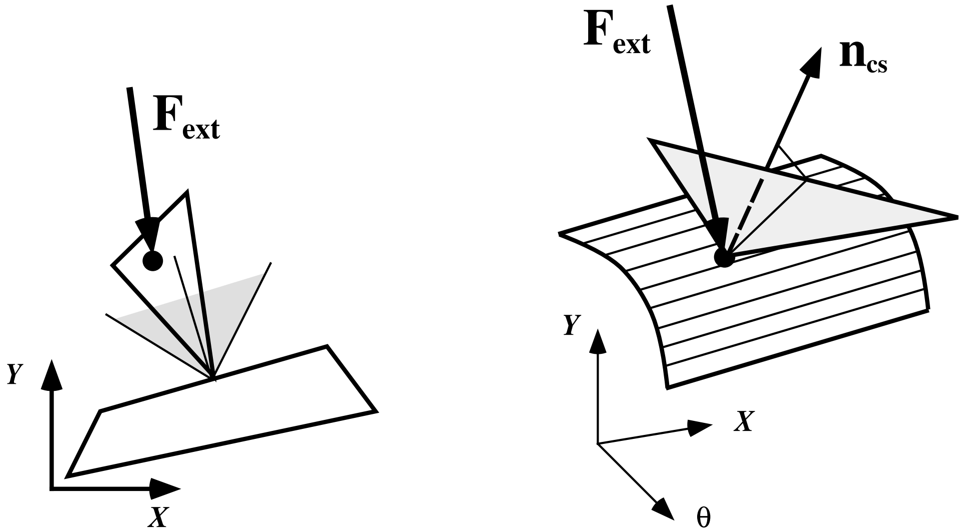


Figure 2.6: Coulomb friction represented as a friction cone, and the corresponding (x, y, θ) configuration space friction cone for a single contact facet.

cone may be viewed as a projection of the configuration space friction cone into the (x, y) plane. The geometric interpretation of the configuration space friction cone as the range of possible reaction forces and torques due to an applied force remains unchanged.

Figure 2.6 illustrates the cartesian and configuration space friction cones for a single (type B) contact represented by a CS facet. Friction cones for contacts with edges or vertices on the CS corresponding to multiple object feature contacts are derived by computing the set sum, or Minkowski[4] sum, for the friction cones of each of the individual CS facets involved at the point of contact. The equations and methods used to construct the configuration space friction cones for each of the above contact cases are discussed in detail in Section 5.4.2.1. The important point to keep in mind here is the interpretation of the friction cone as a geometric representation of the set of possible reaction forces at a point on the surface of the CS.

Elastic and Inelastic Collisions

Collisions between moving objects are another form of interaction that produce reaction forces which in turn determine the resulting motion. In the simplest case involving direct collisions between two objects represented as point masses, the col-

[4]The set sum is defined as: $A \oplus B = \{\overline{a} + \overline{b} | \overline{a} \in A, \overline{b} \in B\}$.

lision event may be divided into two stages: *deformation* and *restitution*.[5]

The ratio of the restitution impulse over the deformation impulse (Poisson's hypothesis) is the coefficient of restitution:

$$e = \frac{\int Rdt}{\int Pdt}$$

where $\int Pdt$ is the total impulse over time during deformation, $\int Rdt$ is the total impulse over time during restitution, and e has a value somewhere between 0 and 1. If $e = 1$ then the collision is perfectly elastic and energy as well as momentum are conserved during the collision. The coefficient of restitution depends to a large extent on the materials of the two objects, but may also depend on the velocities involved, as well as the shapes and sizes of the colliding objects. For the simple case described above, we may take e to be a constant and use it to compute the normal velocities of the colliding objects immediately after collision using:

$$e = \frac{v'_B - v'_A}{v_A - v_B}. \tag{2.1}$$

We should note that in general the nature of collisions can be considerably more complex than for the case described above (see Wang [76]). Nevertheless, we may use equation 2.1 as a rough approximation to the actual behavior of colliding objects for the purposes of bounding the range of possible motions involving collisions.

2.4.2 Forward Projections

As noted earlier, we are interested in determining where the moving object *will go*, or at least in further constraining where it *might go*, under a given set of conditions. What we have developed up to this point is the complete set of kinematic constraints given by the CS, as well as models of interaction mechanics covering frictional sliding and inelastic collisions. What remains is to combine these with the appropriate initial conditions to construct the set of possible motions. In particular, from a specified set of initial conditions we will construct the motion or set of motions defined by the *forward projection*:

$$S_2 = F_A(S_1) \tag{2.2}$$

where S_1 is a point, set of points, or a region in configuration space (state space), A is a (possibly varying) force applied to the moving object, and S_2 is the point, set of points, or region in configuration space traversed by the reference point of the moving object.[6] An important point to keep in mind is that the forward projection

[5]By direct we mean that the velocities are collinear. In examples where the collision is oblique, we refer to the components of the two velocities that are collinear.

[6]The notation for A is derived from the term **Action** from the field of robot motion planning, from which the definition of the forward projection is taken (see Erdmann [28]).

does not necessarily describe the *exact* motion of the moving object. In fact, only when we have a known starting point S_1, a known force A, and detailed mechanics models for the interactions with the surface of the CS, will we be able to generate an exact trajectory through configuration space for the object's motion. In general, S_1 will be a bounded *region* in configuration space, A will either be an exact or bounded function, and the models for the mechanics of interaction will be approximate. As a result, S_2 will itself be a region in configuration space which will contain, as a subset, the exact motion of the object. In those cases where we may have a number of initial positions, or a set of discrete approximations to a region containing possible initial conditions, we may compute the overall forward projection as the *union* of individual forward projections, or:

$$F_A\left(\bigcup_i S_i\right) = \bigcup_i F_A\left(S_i\right) \tag{2.3}$$

where i is the index to a particular subset of initial conditions. This will be particularly useful later on when we will implement forward projections by means of numerical integration or geometrical construction.

The above description of a forward projection seems strikingly similar to the definition of a simulation. In fact, we may view the forward projection as a *superset* of simulations. Again, whereas a simulation describes where a system *will go* for a given set of parameters, the forward projection bounds where a system *can* and *cannot go*, and is thus considered as an extension of the motion constraint set originating with the CS. The one piece of information not included in the forward projection that is available in a simulation is the time history of a motion since we do not consider velocities in our state space.

We will now briefly consider two broad classes of forward projection, one depending on a relatively detailed model of local interaction, and the other relying on less exact but more general global energy constraints. In the discussion that follows, we will assume that the values of the parameters mentioned are known exactly unless otherwise specified. Although the construction of forward projections that incorporate an explicit representation of parametric uncertainty is an important aspect of robust motion analysis, we will not consider such representations here due to the additional level of complexity that doing so would introduce. Where possible, we shall adopt conservative approximations in an attempt to overcome the limitations of not considering uncertainty explicitly. In the next chapter we will discuss a number of tools and representations that may be used to examine some of the effects of uncertainty. An excellent treatment of bounded parametric uncertainty applied to motion analysis is given by Brost [13].

Numerical Simulation

Given a discrete initial position, or set of positions, along with a relatively detailed model of the mechanics of interaction between objects, the most direct means for determining where an object will go is simulation via numerical integration. For example, the motion of a compliantly held rigid part during an assembly operation may be computed using a quasi-static model of dynamics where forces due to spring displacements and Coulomb friction dominate those due to velocities and accelerations (see Whitney [81]). Briefly, a motion path may be integrated as follows. At each point in the integration, the force equilibrium at a contact point is computed. For a quasi-static model, the reaction force at the contact point is assumed to lie on the edge of the friction cone such that the net force is approximately zero. Because the reaction force is at one extreme of the range of forces generated in response to an applied force, the moving object is in a state of *impending motion* in the direction of the applied force projected into the plane tangent to the CS surface at the point of contact. A small incremental displacement is made along the surface in this direction, and the integration continues from the new position.

Where available, more complete dynamic models including damping and inertial effects may be included to integrate more detailed and accurate trajectories. In those cases where initial position or other parameters are not known precisely, an approximation to the forward projection may be generated by numerically integrating paths corresponding to extremes of parameter values and bundling these paths together to form the union.[7]

Energy Bounds for Conservative Systems

Often it will be the case that generating a detailed simulation of object motion will be impractical or even impossible given the amount of information available about the system and the characteristics of the interactions. In such cases, we may rely on simpler, more conservative, approximations of object motions to provide an upper bound on the range of possible motions contained by the forward projection. One such bound due to Brost [13] limits the possible motions of an object dropped from rest in a gravity field using conservation of energy arguments. Specifically, Brost noted that an object, represented by a point in an (x, y, θ) configuration space, could not reach any other point in configuration space that had a higher relative potential energy, no matter what (conservative) interactions took place. This bound may be expressed geometrically as a plane through the (x, y, θ) dropping point and

[7] Another approach due to Brost [13] is to directly integrate motion ranges at discrete points on the surface of the CS and connect these to form boundaries of reachable regions. These bounded regions are then "lifted" from the surface of the CS into the free configuration space to form bounded "volumes". In Brost's application, these regions correspond to *backprojections* from goal regions, although the same mechanism may be applied to forward projections as well. See Brost [13].

perpendicular to the gravity vector which partitions the configuration space into two half spaces representing reachable and unreachable configurations. The intersection of the half space of reachable configurations and the free space bounded by the CS presents a simple means of constructing a very conservative forward projection. In a particularly elegant metaphor for interpreting the meaning of this forward projection, Brost likens the resulting intersections to "puddles" of water on the surface of the CS.[8]

Recalling our earlier discussion on impacts and coefficients of restitution, we may interpret the plane bounding the maximum potential energy as the forward projection assuming perfectly elastic collisions with $e = 1$. Since most materials have some internal damping, we would like to examine forward projections for cases where $e < 1$, i.e. where collisions are assumed to be inelastic. Let us assume that the stationary object (B) remains stationary throughout an impact, then $v_B = v'_B = 0$ and:

$$v'_A = -ev_A \tag{2.4}$$

where the minus sign indicates that the post-impact velocity of the moving object (A) is in the opposite direction to the pre-impact velocity.

Let us consider a simple example with the purpose of obtaining an approximate bound to the forward projection for a dropped object based on a value of $e < 1$.[9] In Figure 2.7 we have a particle being dropped from rest in a gravity field onto a (frictionless) flat surface patch tilted by an amount ψ_1 relative to the horizontal. A second surface patch is positioned horizontally at the same height as the first such that the particle's parabolic trajectory after the first impact intersects the second patch, and is oriented by an amount ψ_2 such that the particle's motion after the second impact is vertical, as shown in Figure 2.7. This particular configuration of two consecutive impacts was chosen to allow us to explore the range of (x, y) excursions of a particle with the minimum number of collisions (two) necessary to raise the particle back to its maximum height with zero horizontal velocity (i.e. maximum potential energy). Since the position and orientation of the second patch is dependent on the orientation of the first patch, we may explore the range of particle motions by varying only one parameter ψ_1. The lower half of Figure 2.7 plots the locus of points, for various values of ψ_1 from $0 \rightarrow \frac{\pi}{8}$, of *(i)* the maximum height h_{max1} reached by the particle after impacting the first angled surface, and *(ii)* the maximum height h_{max2} reached after the second impact.

[8]Actually, the puddle metaphor is particularly useful for Brost's application of constructing a *backprojection* from a goal state by slowly "filling" the puddle from a goal state until it either reaches a constraint or overflows into another region on the CS surface. The volume enclosed by the puddle, then, represents the set of (x, y, θ) configurations from which an object may be dropped and still be guaranteed to reach the desired goal.

[9]Derivations for the models used in this and the following examples are presented in Appendix A.1.

The shaded region in the lower half of Figure 2.7 represents a conservative bound on the (x, y) positions that the particle may reach or pass through for any pair of collisions. This region corresponds to a simple conservative model of the forward projection for the dropped particle. Since each semi-elastic collision involves a loss of energy, we postulate that this region contains all of the positions reachable by a dropped particle for any number of collisions. Figure 2.8 shows a number of double bounce maximum height curves for various values of e. We notice that, as expected, the lower the value of e, the smaller the forward projection becomes. Again, we may conservatively approximate the boundaries of the forward projections for a dropped particle that may undergo an arbitrary number of collisions with randomly oriented surfaces by straight lines as shown in Figure 2.8. By symmetry about the vertical axis, the forward projection for a dropped particle with a maximum coefficient of restitution e becomes a *cone* in (x, y), as shown in the bottom of Figure 2.8.

We have so far neglected the role of friction and rotational motions in the forward projection of a dropped object, modeled here as a particle. We expect that the addition of friction into the above models will serve to further tighten the bounds on the forward projection. Also, since energy is present in rotational as well as translational motion, we expect similar bounds to extend into the θ dimension of configuration space where a point represents the position and orientation of a rigid object.[10] As noted earlier, the nature of impacts for rigid objects in the presence of friction can be considerably more complex than for a particle (see Wang [76]). Nevertheless, approximate models for such interactions do exist, and the above examples serve to illustrate the nature of conservative models for forward projections of objects undergoing motions that are potentially far more complex.

Energy Bounds for Non-Conservative Systems

One more forward projection model that we will briefly consider applies to the case where an object interacts with a surface undergoing a forced vibration. We mention this case here because it is representative of a number of real world examples, and because bounds on the forward projection are particularly useful in describing the behavior of such systems since they are known to be chaotic in general.[11]

We once again begin with a simple model consisting of a particle that is either dropped or placed onto a surface that is undergoing a forced oscillation of $y(t) =$

[10] One important exception noted by Brost [12] has to do with *rolling* motions of objects. If an object is allowed to roll, it is theoretically possible that the forward projection could extend to infinity for interactions with a flat surface. Since we are concerned with objects represented as polygons, we do not consider this exception since any rolling motion of a polygonal object will involve impacts between the vertices of the moving object and the surface on which rolling takes place. These impacts will remove energy from the moving object and eventually bring it to rest.

[11] For a classic example, see "The Dynamics of a Bouncing Ball", Section 2.4 in Guckenheimer et. al. [36].

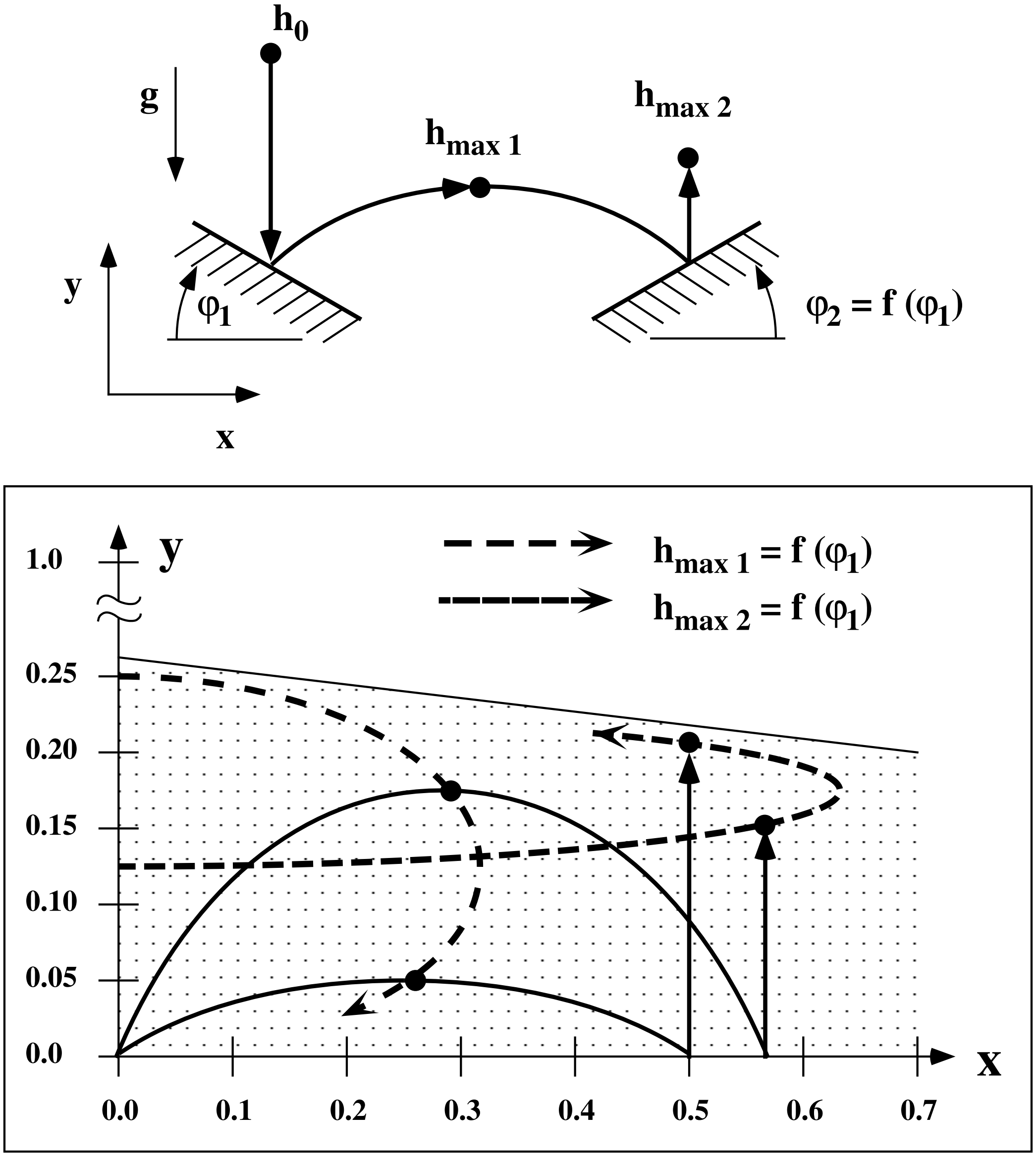
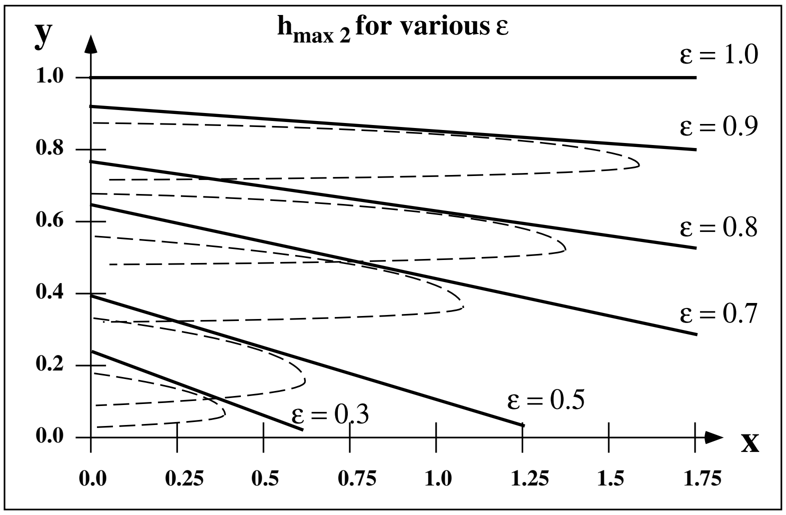


Figure 2.7: Locus of positions of maximum height reached by a particle after impacting two surfaces as the orientation ψ of the first surface is varied from $0 \rightarrow \frac{\pi}{8}$.



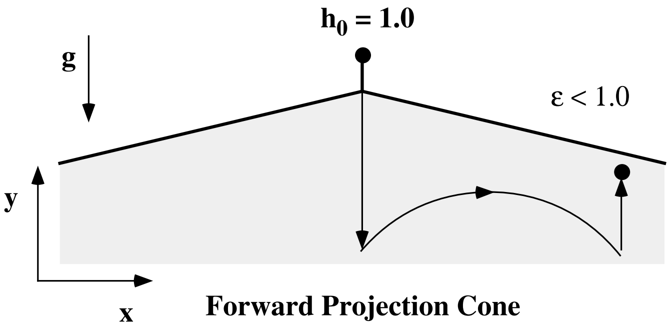


Figure 2.8: Locus of double bounce maximum height positions for selected values of e.

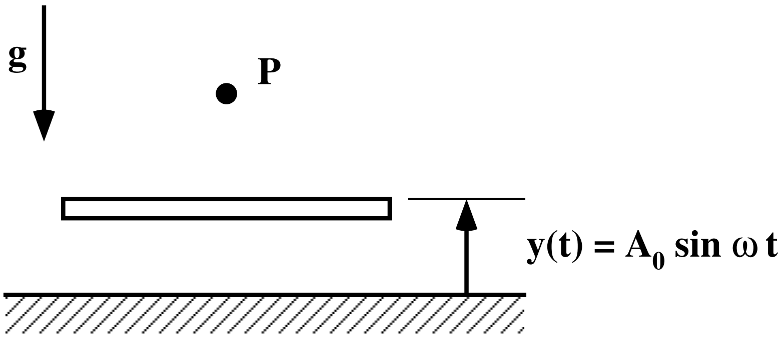


Figure 2.9: A particle released onto a vibrating table in a gravity field.

$A_0 \sin(\omega t)$, as shown in Figure 2.9. Figure 2.10 shows the results of a numerical simulation of the maximum height achieved after each bounce of the particle. Because energy is being put into the system by the table vibration, the conservation of energy arguments made earlier do not apply here.

Although the system is not conservative, we can establish bounds on the maximum amount of energy that may be imparted into the particle in any one bounce. Furthermore, if we assume that the collision between the particle and table is inelastic ($e < 1$), then each impact will also remove energy from the particle. Given these two observations, we may establish an equilibrium scenario where the maximum amount of energy that may be imparted to the particle by the table exactly balances the energy lost in the impact with the table. The derivation of this bound is given in Appendix A.2, and the resulting expression for the maximum height that may be reached by a particle starting from rest is:

$$H_{max} = \frac{(A_0\omega)^2}{2g}\frac{(1+e)^2}{(1-e)^2} \tag{2.5}$$

It turns out that in practice, the maximum height given by H_{max} is very conservative. For example, taking a histogram of the simulation shown in Figure 2.10 after 1000 impacts, 100% of the maximum bounce heights were less than 10% of the estimated H_{max}.[12]

We could use the result of Equation 2.5 to construct a conservative approximation to the forward projection of an object impacting a vibrating surface. Figure 2.11 illustrates the range of post-impact velocities for a particle impacting a vibrating oriented surface as a function of the range of possible pre-impact velocities. In configuration space, the forward projection of the post-impact velocity ranges from

[12]We notice that if the collision is elastic ($e = 1$) then H_{max} as given by Equation 2.5 is infinite.

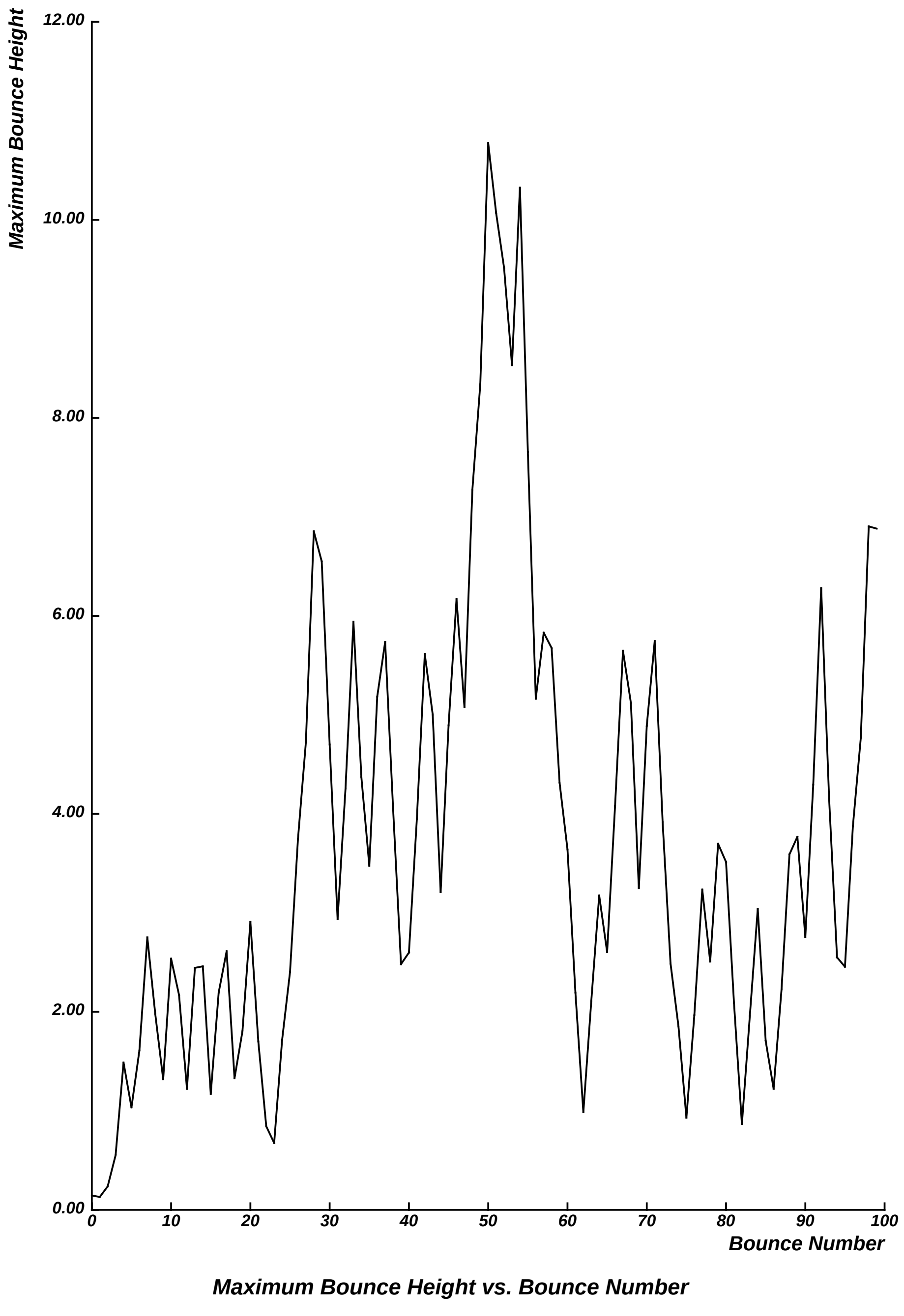


Figure 2.10: Numerical simulation of the maximum height achieved after each bounce of a particle on a vibrating table, vs. bounce number.

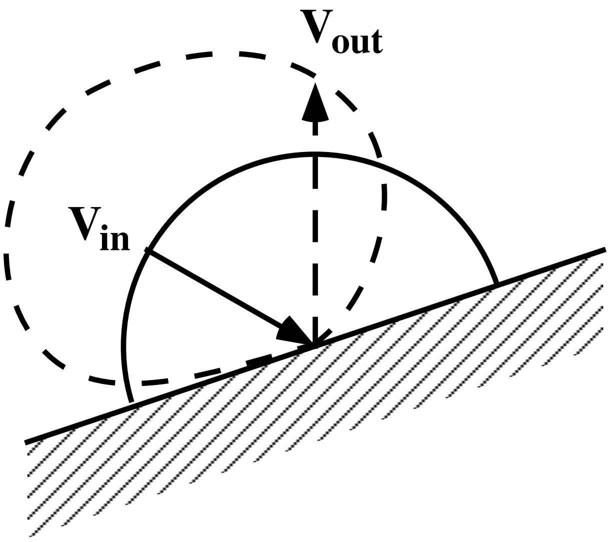


Figure 2.11: Illustration of the range of post-impact velocities corresponding to a pre-impact velocity range.

the point of impact would form a "bubble" on the surface of the CS at that point. For a given starting point on the surface of the CS, the forward projection of an object's motion could be constructed by generating the forward projection bubble from that point, and then expanding outward from the boundary of the bubble by using points on the boundary that intersect the CS as starting points for generating new bubbles. The process would stop when, at every point on the boundary of the expanded forward projection, the post-impact velocity range points into the expanded forward projection. Of course, there is no guarantee that the forward projection operation would terminate since the forward projection could be unbounded. Consider, for example, the case of a vibrating staircase where the local point forward projection on one step is just large enough to enclose the edge of the next step, making it possible for an object to "hop" up the staircase step by step, even though the input vibration may be small.

2.4.3 Support Constraints

Forward projections act to further constrain the regions of configuration space, bounded by the kinematic constraints, in which object motions may occur. One important class of motions has to do with the concept of *support*. Up to now we have assumed that the moving objects in our representations are constrained to move in the (x, y) plane without explicitly considering the nature of this constraint.

Consider the following experiment. Place an object on a flat table, then slowly rotate the object out of the plane of the table surface while keeping one vertex or edge of the object in contact with the surface at all times. If the object started in a stable rest orientation on the table, then any motion of the object away from the table surface will act to raise the center of mass of the object relative to the stable rest position. In other words, the table surface supports the object in a stable configuration so that the object's center of mass, and hence its potential energy, is at a local minimum. If the same object were placed with its center of gravity over the edge of the table, it would be possible to rotate the object in contact with the table edge in such a way that the object's center of gravity would be lowered relative to it's initial configuration. If the object were released from this configuration in the presence of gravity, it would naturally tend towards the lower energy state and fall off the table.

This simple experiment serves to illustrate the nature of the support constraint in terms of potential energy. Viewed another way, if we recall the energy bounds used to generate forward projections earlier, we can classify support as the set of configurations whose forward projections under gravity are constrained to lie within the plane. The reason for taking this view of support is that we can use the same techniques developed above to generate the *boundaries* dividing supported and unsupported regions of (x, y, θ) configuration space. Figure 2.12 shows a moving object modeled as a planar polygon lying on top of a stationary planar polygon, together with an illustration of a surface in configuration space representing the boundary between supported and unsupported configurations of the moving object relative to the stationary supporting object. This surface is similar to the CS in that it partitions the configuration space into two distinct regions: supported and unsupported. The surface does *not* represent contact constraints between objects, but rather the set of points in configuration space at which the moving object's support status *transitions* from supported to unsupported. Specifically, points inside the bounding surface represent (x, y, θ) configurations of the moving polygon that are supported by the stationary supporting polygon, whereas points outside the surface represent configurations where the moving polygon is unsupported and would fall out of the (x, y) plane.

The structure of the support constraint boundaries is in many ways much simpler than that for contact constraints, but there are also a few "interesting" and more complex cases that appear quite often. For the example shown in Figure 2.12 we notice that many of the support constraint surfaces are flat planes parallel to the θ axis of the configuration space. These surfaces are constructed by mapping the points where the center of gravity of the moving polygon crosses an edge of the support polygon. If the reference point of the moving polygon used to construct the configuration space is coincident with the polygon's center of gravity, then the support transition boundary is simply that edge of the support polygon swept from

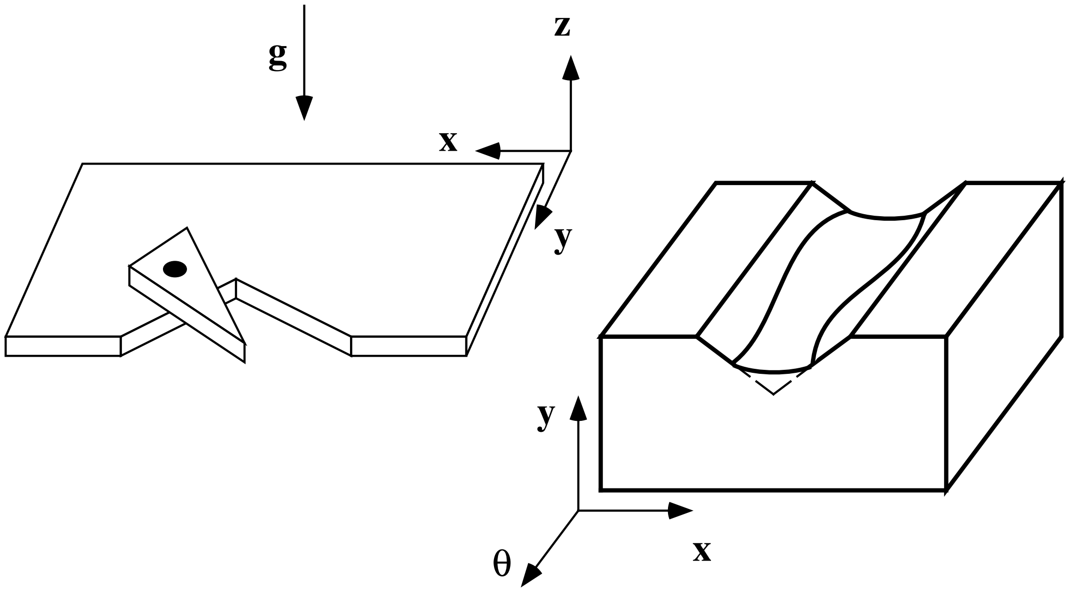


Figure 2.12: A moving planar polygon supported by a stationary planar polygon, and the corresponding support transition boundaries in (x, y, θ) configuration space.

$0 \rightarrow 2\pi$ along the θ-axis of the configuration space. We note, however, that a number of the support constraint surfaces shown are curved, not flat, in the (x, y, θ) dimensions. These surfaces also correspond to configurations where the moving polygon's support status is in transition from supported to unsupported, but the polygon's center of gravity lies outside of the supporting polygon's contour. Such cases may occur within concave contours of the support polygon where the moving polygon's center of gravity is contained within the *convex hull* of the intersection of the moving and supporting polygons.[13] Section 5.5.2 contains a more detailed discussion of support transition boundaries and how to compute them. At this point we are more concerned with the qualitative interpretation of support constraint boundaries, and their relationship to contact facets and forward projections as an additional form of motion constraint.

One final note regarding support constraint boundaries has to do with their relationship to higher dimensional motions whose description is beyond the scope of (x, y, θ) configuration space. In particular, the motion of an object falling off of a planar support surface cannot be represented solely in terms of (x, y, θ) as the number of

[13]One other possible scenario involves moving polygons whose centers of gravity are outside of their contours. In such cases, it is possible to have the **cg** inside the contour of the supporting polygon but still have the moving polygon be unsupported.

degrees of freedom for such a motion, and hence the number of position parameters required to characterize it, are greater than three. The support constraint boundaries just described represent those configurations where a planar motion represented as a trajectory in (x, y, θ) would *transition* to a trajectory in a higher dimensional configuration space of which (x, y, θ) is a subset. The support constraint boundaries, then, represent a lower dimensional mapping of the transitions to higher dimensional motions. This transition model constitutes a tradeoff where we sacrifice detailed knowledge of these higher dimensional motions in return for a simpler and more manageable representation. In the next section we will examine further the use of such simplifications that allow us to reduce the dimensionality and complexity of motion constraints represented in configuration space, as well as explore ways of representing the different forms of motion constraints discussed above within the same global framework.

2.5 Mapping Constraints into Motion Space

The CS, forward projections, and support constraints all partition the configuration space into regions that are either reachable or unreachable with respect to motions of the moving object. Our purpose in generating these constraints is to combine them in such a way as to predict the behavior, expressed as motions, of the system under study. In this section we will briefly discuss some of the ways of combining the geometric representations of motion constraint developed above within configuration space so as to make such behavior explicit. The important point to keep in mind as we manipulate the various different forms of constraint representations is that they are all expressed in terms of the **motion** of the moving object as captured by the configuration space. The factors that determine these constraints: shape, mechanics and dynamics, are all represented *implicitly* in the constraints – *it is the constraints on the motions themselves that are explicit.*

Contact Constraints – The CS

We have already discussed the representation of contact constraints in configuration space. By taking the union of contact facets for all feature pairs for a moving and stationary object, we generate a representation of the entire set of kinematically distinct contact interactions for those objects that also serves to partition the configuration space into regions of reachable and unreachable positions of the moving object.

Forward Projections

The two broad classes of forward projections discussed in Section 2.4.2, i.e. exact and energy bounded, may be represented in configuration space as one-dimensional

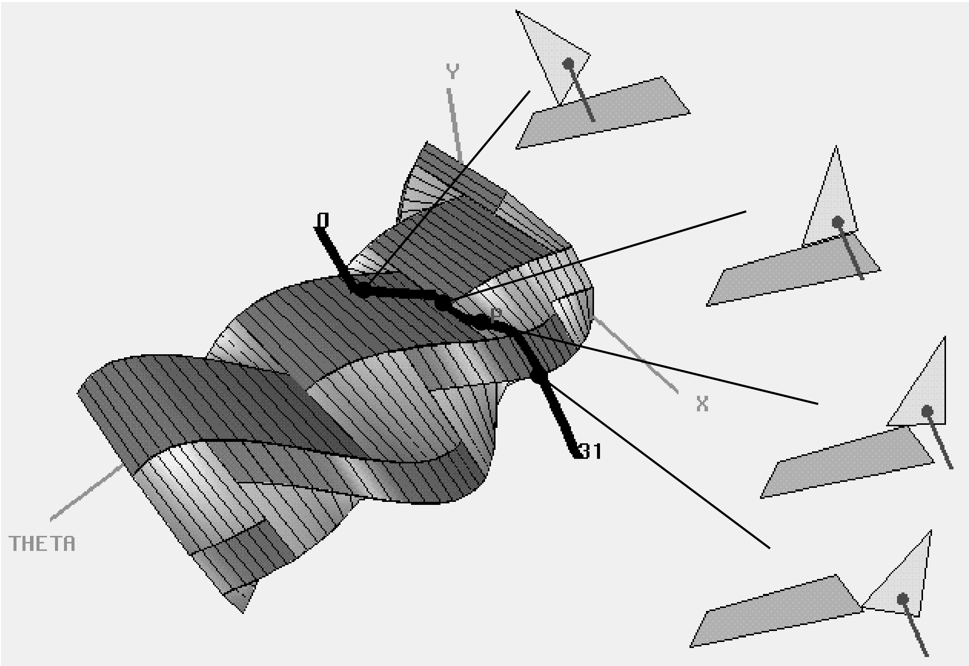


Figure 2.13: A forward projection generated from a numerically integrated motion path across the surface of a CS.

space curves and three-dimensional bounded regions, respectively. Figure 2.13 illustrates an exact trajectory for a moving object, starting from a specified initial position, represented as a space curve on the surface of the CS of Figure 2.4. The forward projection consists of the set of points along the curve.[14] In the case of energy bounded motion dynamics, a flat plane (for $e = 1$) perpendicular to the direction of gravity would intersect the surface of the CS. The forward projection would then consist of the set of points corresponding to the volume of configuration space bounded below by the surface of the CS and above by the bounded energy plane.

Intersection of Contact and Support Constraints

A further simplification for the support constraint regions introduced in Section 2.4.3, both in terms of computation and representation, is to directly compute the *intersec-*

[14]Although rarely represented in terms of boundaries, we may consider a trajectory generated from exact integration of motion mechanics to be a tubular region in configuration space bounded by motion constraints that have been collapsed onto a one-dimensional space curve. This view is semantically consistent with the much looser bounds imposed by the constraint surfaces generated for an energy bounded forward projection.

tion of the support constraint boundaries with the surface of the CS in configuration space. The result is a partitioning of the **surface** of the CS into configurations where the moving object is: (*i*) in contact with the outer contour of a stationary object in the same plane, and (*ii*) either supported or unsupported by a second stationary object *underneath* the plane of motion. This simplification allows us to represent the subset of support transitions that occur while an object is in contact with another object in the same plane, without obscuring the display of the CS by trying to represent another constraint surface in configuration space. Figure 2.14 illustrates such a partitioning of the CS surface, where the darkened regions represent contact configurations that are unsupported. Since most of the motions we shall be concerned with are constrained motions where the moving object is in contact with the stationary object, this simplified representation of support constraints preserves most of the useful information present in the general support constraint boundary surface.

Superposition of Configuration Space Slices

As we mentioned in Section 2.2, we will deal primarily with interactions between objects constrained to have three or fewer degrees of freedom. It is important to note that this constraint does not require that the objects we consider must themselves be planar. Indeed, with a (not insignificant) amount of additional work and computation, it is possible to represent interactions between three dimensional polyhedra. Another approximate but more convenient approach is to model three dimensional objects as a series of polygons corresponding to slices of the objects at different heights. The $2 + 1D$ CS for each of these slices may then be generated using the tools and representations described earlier in this chapter and then *superimposed* within the same (x, y, θ) configuration space. The resulting composite CS structure, although somewhat more complex and with a much larger number of contact facets (many of which would be occluded by other facets), could be used for analysis in much the same manner as the CS for a single set of polygon interactions.[15]

Other Constraint Mappings

There are a number of other mappings of constraints to configuration space that, depending on the application, can be useful in representing function in terms of motion. The following three constraint mappings are due to Brost [13] and were

[15]One caveat to this approach has to do with the fact that interactions with the CS surface that correspond to contacts among slices at different heights would impose out of plane torques to the object that would be outside the scope of the (x, y, θ) representation. Special care would have to be taken to ensure that the distribution of forces among the slices in contact was consistent. Techniques analogous to the support transition boundaries of Section 2.4.3 might also be useful in representing the effects of these torques.

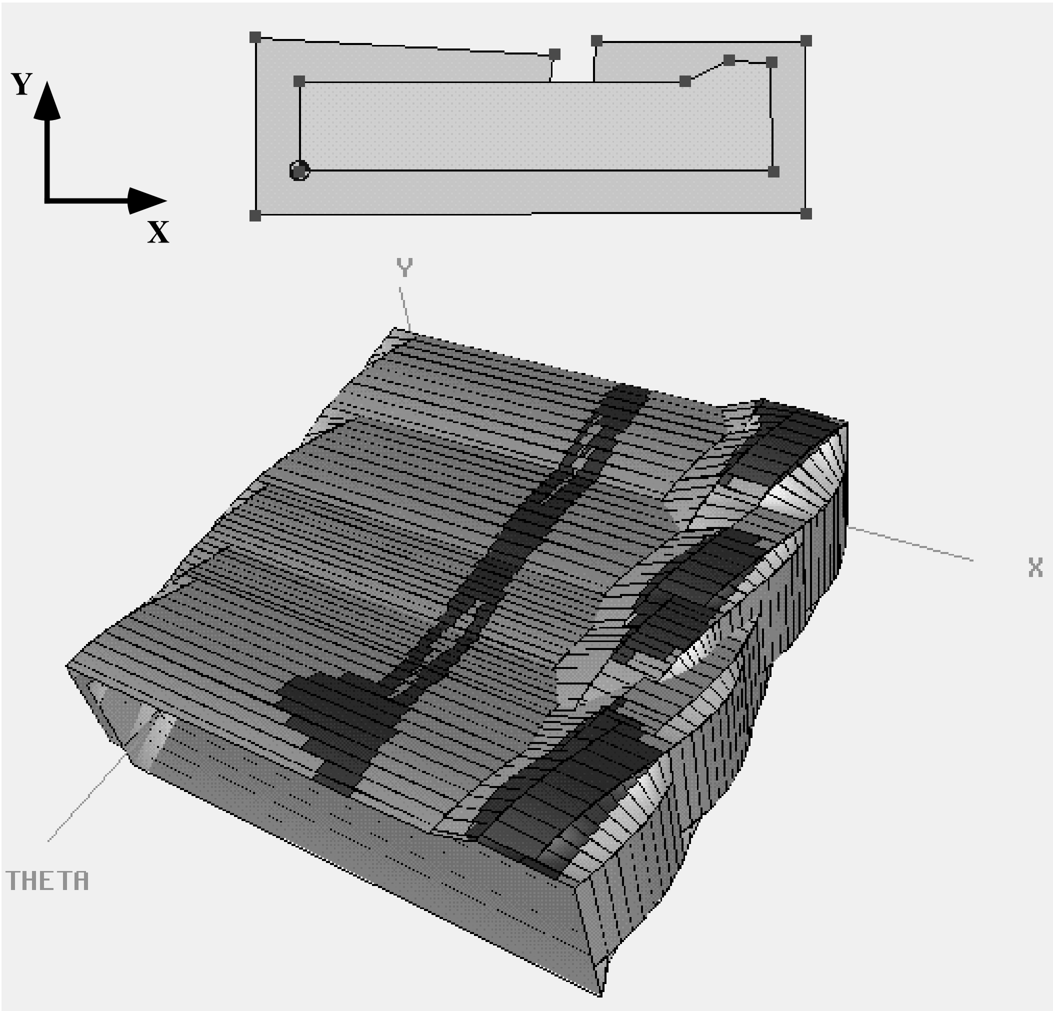


Figure 2.14: Support transition boundaries intersected with the surface of the CS.

applied to the tasks of analyzing and planning pushing and dropping motions of planar polygons:

- **Sticking regions due to friction.** Given a constant applied force, the contact configurations on the surface of the CS that would result in no motion due to friction are identified and marked so that they may be avoided in constructing backprojections from a goal configuration.

- **State transition cones.** Given an applied force, local forward projections may be generated and displayed at discrete points on the surface of the CS to indicate the "flow" field of constrained motions. This representation is particularly useful in capturing and representing uncertainty in motion parameters in terms of cones of possible motions from each discrete point.

- **Arbitrary Constraints.** Some object features should be avoided during a manipulation operation because they are particularly delicate, or may be coated with an adhesive or other material which should not be brought into contact with other objects except in a certain predefined configuration. The constraint facets corresponding to contacts with these features are labeled as *off limits*.

Some other potentially useful constraint mappings derived from the energy bounded forward projections discussed earlier include:

- **Topographic potential energy map.** The plane corresponding to an $e = 1$ bounded energy forward projection represents a constraint for a single energy level. It is not difficult to imagine generating curves on the CS surface corresponding to slices of the CS at different energy levels. Such a family of contours would be equivalent to a topographical map of the CS surface, and would provide a global picture of the "hills and valleys" in the set of motion constraints. The "valleys" in particular are interesting to us since they correspond to local minima in which the moving object could come to rest for certain motions.

- **CS intersections with vibratory impact forward projection boundaries.** Similar to the topographical boundaries in the previous example, curves representing the set of configurations where the vibratory forward projection boundaries intersect the CS surface serve to partition the CS surface into reachable and unreachable configurations for a given set of initial conditions. As the amplitude or frequency of vibration is varied, these boundaries would give a global picture of the changes in system behavior accompanying changes in these parameters. As in the case of the support constraint boundaries, the resulting intersection curves would not obscure surface details of the CS.

The above representations in (x, y, θ) configuration space capture motion constraints in terms of geometric structures that include parametric surfaces, planes, and space curves. These structures are useful both as computationally accessible constraint representations suitable for manipulation by algorithms, and as visual representations for display and interpretation by humans. This second application is particularly attractive since we will need to understand the nature of the constraints we wish to impose before we can attempt to develop algorithms that generate them automatically.

2.6 Summary

In this chapter we explored the representation of function in terms of motion constraints. We examined what we mean when we speak of functional constraints on motion, and underscored the need for a mathematically precise representation for these constraints within configuration space. We described a zero-velocity state space (configuration space) to capture object motions, and in so doing focus our attention on motion instead of shape as a language in which to describe the functionality of object interactions. Since, in general, the configuration space can be quite large, we limited our discussion to objects whose motions were constrained to lie in a plane. The result was a three dimensional configuration space whose axes are (x, y, θ). We considered two broad classes of motion constraints: kinematic and non-kinematic. Kinematic motion constraints arise from interactions between object shapes, and may take the form of individual contact surfaces or supersets of contact constraints in configuration space. Non-kinematic motion constraints arise from the forces derived from the mechanics of contact, as well as externally applied forces and gravity fields. The mechanics of contact we considered included sliding friction, represented geometrically as the friction cone in configuration space, as well as elastic and inelastic collisions between objects. From these mechanics we were able to construct forward projections of motions that further partitioned configuration space into regions of reachable and unreachable states. Two kinds of forward projection that we considered in detail were the exact integration of motions for the cases where we had a detailed model of the dynamics, and bounded energy motion constraints for those cases where the dynamics could not easily be characterized. In both cases, we likened forward projections to a timeless superset of simulations of object motions under the specified constraints. A third special case of forward projection we considered was the support constraint, the stability of which was viewed as a constraint on the potential energy in a gravity field of the moving object's center of gravity while resting on a flat surface. Those regions of configuration space where the potential energy of the object could be reduced by means of a rotation out of the plane were deemed to be unsupported. The support constraint was also given as an example of a simplification whereby constraints on higher dimensional motions could be represented in a lower dimensional configuration space as transitions between planar and non-planar motions. Finally, after characterizing the above set of motion constraints, we examined various means by which those constraints could be combined within the configuration space representation.

The purpose of this chapter was to develop the representations that will serve as the foundation upon which we may build a set of tools that will allow us to perform both analysis and design of functionally useful shapes. To make these representations and visualization techniques more concrete, in the next chapter we will introduce a set of four examples: peg-in-hole assembly, vibratory bowl feeders, assembly pallets and fixtures, and another vibratory feeder known as APOS. These examples have been chosen because they span the set of constraint representations developed here, as well as to highlight similarities among and differences between the various forms of functional constraints.

Visualization and Application Domains

Chapter 3

In this chapter we will examine four application domains introduced in Figure 1.8: compliant assembly, vibratory bowl feeders, assembly fixtures, and the APOS vibratory feeding system. We will use the motion constraint representations developed in the previous chapter to visualize, reason about and analyze the functional characteristics of examples from each of these domains in terms of motion constraints.

In Section 2.3.1 we referred to the similarities between the surface of a CS facet and a "real" surface that produced reaction forces and torques in response to applied forces. In Section 2.3.2 we referred to features on the surface of the CS using terms such as valleys, ridges, and peaks that convey images of multiple features combining to form what amounts to a landscape in configuration space. The intent of this visual imagery is to convey an intuitive feel for some of the structure imbedded in the CS and how these constraints act to guide motions of a point representing the motion of a physical object.

A point in configuration space, whose (x, y) components correspond to the position of the reference point of, and the θ component to the orientation of, the moving object can be thought of as the point of action through which external and reaction forces act to constrain the motion of the object. All interactions between shapes of both the moving and stationary objects are combined so as to be local to this point. If we imagine that point as a ball bouncing or sliding across the surface of the CS constraints, then we have a powerful metaphor with which to visualize how constraints interact. For example, we have discussed energy in terms of non-kinematic constraints, or bounds, that determine where a point, or ball, may travel in the presence of an externally applied motive force such as weight due to gravity. This energy imposes on the configuration space a sense of up and down that immediately implies a sense of where the ball will and will not go based on its interaction with the CS surface. The curved surfaces of individual CS facets guide the motion of the ball along curved trajectories imposed by the ever-changing surface normal along the facet, while valleys between CS facets guide and constrain the ball along their floor. Intuition regarding the behavior of rolling and bouncing balls can prove quite useful in the more abstract domains of mathematical constraint surfaces and configuration space.

With these visual metaphors for constraint surfaces in mind, we return to the question of what to do with them – how do we represent function? First of all, forward projections, like simulations, provide a visual verification of motion in the presence of motion constraints. With the configuration space representation, however, we have in addition to a verification of where the motion *will go* a sense of where the motion might have gone, or might go, as a result of perturbations to one or more system parameters. For example, in examining the motion of a discrete path across a facet surface, we also have in the surface of the facet itself the family of potential motions sharing the same contact – we know at a glance where else that motion could and could not go. As another example, consider the case of an energy-bounded forward projection for a dropped object intersecting the surface of the CS. Were we to expand the range of the cone encompassing reachable states, say by increasing the value of the coefficient of restitution e from 0.8 to 0.9 through a change of materials, then we would immediately be able to determine what new regions of configuration space would now be reachable and what new constraints might interact with the new motions. This sort of "what if" visualization of different scenarios is critical for determining the robustness of a system as well as determining what changes or new features might be required of or desirable in a new design.[1]

There is a considerable amount of geometric detail contained within the motion constraints as shown in Figure 2.4, in some ways perhaps too much detail. Since the CS is a mathematically precise embodiment of the complete set of motion constraints generated by two interacting objects, all of the corresponding kinematic constraint information is available. The question becomes, then, how can we recognize and abstract what we need from what is not necessary? Of course, what is and what isn't necessary depends on the application in question. If we wish to verify a motion or set of motions, as described above, then the detailed quantitative information contained in the CS may be necessary. If, on the other hand, our goal is to abstract functional characteristics for a class of constraints or class of motions across different specific examples, then such detail could prove unnecessary and even distracting. For this purpose, we will develop on *functional metaphors* that seek to describe, in qualitative terms, the topology of the motion constraints (both kinematic and non-kinematic) that best characterize a particular function. We should stress that the role of these metaphors will be to complement, rather than replace, the motion

[1] We refer to the visualization of constraints for a particular set of parameters as *static constraint visualization*. Another form of visualization to be discussed in the next chapter has to do with visualizing the **coupling** between constraints as parameters are varied, which we will refer to as *dynamic constraint visualization*.

constraints described earlier.

Application Domains

The assortment of representations and constraint mappings discussed in the previous sections (contact facets, forward projections and support constraints) allow us to represent a rich set of motion constraints within the (x, y, θ) configuration space. We will now apply various combinations of these representations to a series of specific example domains. In so doing, we hope to make the visualization concepts discussed above more concrete, as well as explore and illustrate the properties and potential usefulness of functional representation using motion constraints. Specifically, we will examine and discuss four example domains where function is derived from motions dominated by shape interactions. The discussion here is intended to be illustrative in nature. In the next chapter we will develop two of the examples in greater detail by implementing a computational environment supporting a set of tools to be used for visualization and design.

3.1 Assembly

Figure 3.1 illustrates what has become a classic instantiation of the assembly problem: the task of inserting a cylindrical peg into a tight clearance hole. Accomplishing this task is complicated by the fact that the position of the peg may not be well known nor the assembly trajectory of the robot precisely controlled due to the presence of uncertainties in position and control. To compensate for positional misalignments, the connection between the peg and the robot incorporates a degree of compliance that is implemented either in hardware, such as a compliant spring device like the Remote Center of Compliance (RCC), or by means of software control of the robot itself [81, 79]. Typical failure modes of an assembly operation include *jamming*, where the forces between the peg and hole are balanced due to friction so that no motion occurs, or *wedging*, where the compliance of the peg itself can result in significant reaction forces between the peg and hole that are independent of any external applied forces and prevent the peg from being moved into or removed from the hole (see Whitney [81]).

Figure 3.2 shows a commonly used representation for axisymmetric parts, where both the peg and hole are modeled as planar polygons. The compliance can be modeled as a generalized spring or generalized damper. For the generalized spring model we have:

$$(\vec{x} - \vec{x_0}) = [C]\vec{F}$$

where the diagonal compliance matrix $[C]$ maps displacements, between the position $\vec{x}$ of a reference point on the peg and the commanded position $/vecx_0$ of that point along a nominal assembly trajectory of the robot, to forces and torques $\vec{F}$ applied to

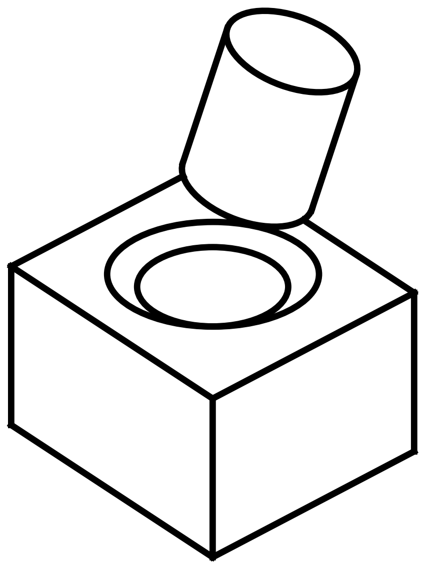


Figure 3.1: The classic peg in hole problem for assembly.

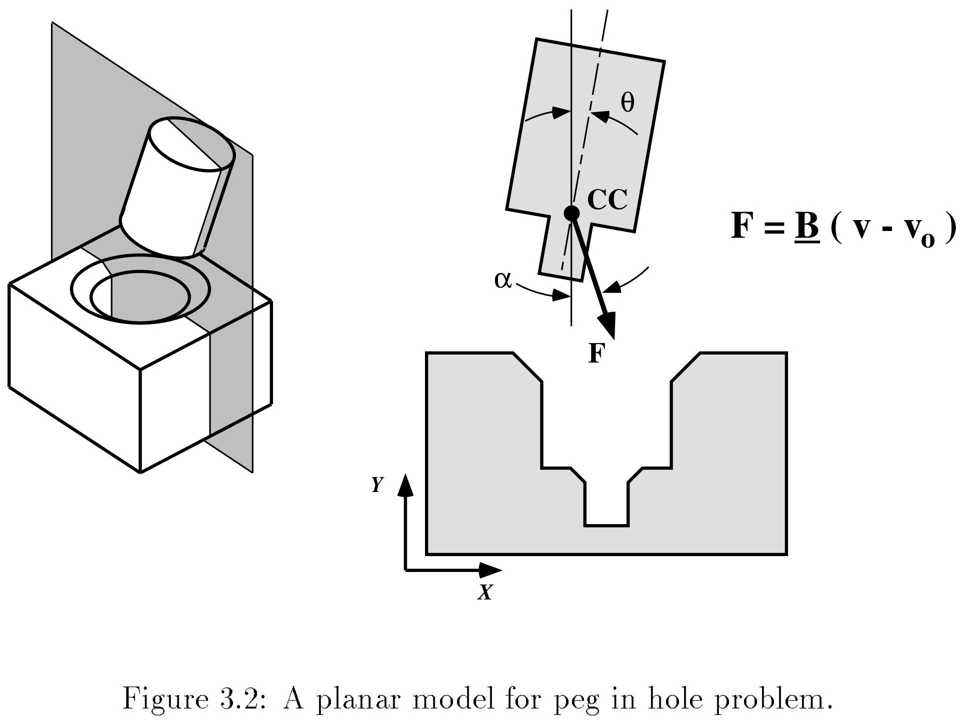


Figure 3.2: A planar model for peg in hole problem.

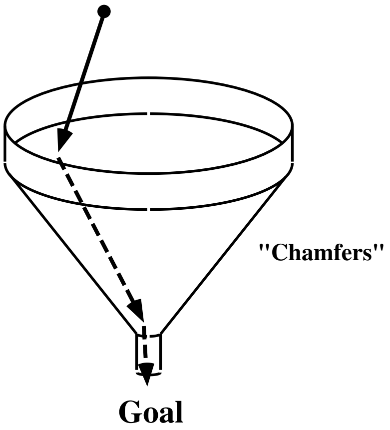


Figure 3.3: Functional metaphor for peg in hole assembly in terms of motion constraints.

the peg at the reference point. The elements of the compliance matrix, as well as the location of the reference point (the compliance center) are important parameters in addition to part shape that must be considered in designing a successful assembly.[2] Compliance based on a generalized damper may be modeled in a similar fashion with displacements replaced by differences in velocity, as shown in Figure 3.2. We use an approximate model of positional uncertainty where motions are integrated from discrete points within a bounded starting region that has been subdivided. The resulting *bundle* of discrete paths provides a crude model of the range of motions possible under uncertainty. Each path may have associated with it a probability that may be used to determine the overall relative reliability of the assembly operation should some of the paths in the bundle fail to reach the goal.

A metaphor for the function embodied in the assembly task in terms of motion constraints is a funnel as shown in Figure 3.3. Essentially, a successful assembly is characterized by a class of motions starting from a set of initial starting configurations that are constrained to reach a single goal state or region due to motion constraint interactions. These motion constraints arise from the kinematic constraints imposed by the contacts between the two part shapes and non-kinematic constraints derived from friction and compliance.

Figure 3.4 shows the configuration representation of a planar peg in hole assembly. We can see that the physical attributes we associate with a hole are retained in the constraints formed by the CS. Specifically, the goal state where the bottom of the peg

[2]As noted earlier, active compliance may be implemented in software, in which case the compliance matrix need not be diagonal (see Schimmels and Peshkin [68]).

is at the bottom of the hole corresponds to a region at the bottom of a corresponding hole in the CS bounded by constraint facets. If we were to take a slice of the CS in the (x, y) plane perpendicular to the θ axis at $\theta = 0$, the width of the CS hole would be equivalent to the *clearance* between the peg and hole; the greater the clearance, the wider the hole. Correspondingly, if the peg were slightly larger than the hole, say for an interference fit, then the hole in the CS would disappear.

With regards to the metaphor of a motion constraint funnel, the most important region on the CS is the entry region immediately surrounding the hole. The facets forming this region are the constraints that will guide assembly motions, whose initial positions and trajectories will vary in the presence of uncertainty, toward the throat of the hole. Once the motions reach the throat of the hole the remaining portions of their trajectories are tightly constrained toward the goal state at the bottom of the hole (Caine [15]).

The idea of letting geometric constraints guide an assembly is a well known strategy identified by a number of researchers[3] and formalized in terms of motion constraints by Mason [54]. A geometrical feature often employed as an aid in assembly is the *chamfer*, which can be viewed as a direct physical instantiation of the motion constraint funnel metaphor. An interesting analog to the chamfer is embodied in the compliant motion strategy of intentionally introducing a rotational and positional offset by tilting the peg relative to the hole before insertion (Inoue [41]). By tilting the peg and placing its lower corner into the hole it is possible to increase the set of initial starting positions from which the peg may be inserted. The result, shown in configuration space in Figure 3.5 **(a)**, is a local entry region on the CS surface that, like the set of constraints for the chamfered hole shown in **(b)**, guides motions toward the throat of the hole. In both cases, an entire range of trajectories is captured and guided by the funnel-like motion constraints, thus improving the overall reliability of the assembly operation. There is one important difference between chamfers and the tilting strategy: straight-line pushing motions alone are not sufficient to perform the tilting strategy since the peg must eventually be aligned with the axis of the hole to eliminate the significant initial angular offset. Hence, the tilting strategy can require a greater degree of complexity in terms of assembly hardware, although passive devices have been developed which extend to this case for simple part geometries (see Draper [25], Caine [15], Strip [72]).

Generally speaking, increasing the size of the funnel-like region of the CS surrounding the hole is desirable in improving assembly reliability. Typically the dimensions of the CS hole itself, i.e. the clearance between the peg and the hole, are fixed by non-assembly design considerations such as maximum allowed slop in a bearing assembly, for example. Parameters that contribute to the motion constraints are the geometrical features of the peg bottom and hole rim, the position of the compliance

[3] See, for example, Simunovic and Whitney [81].

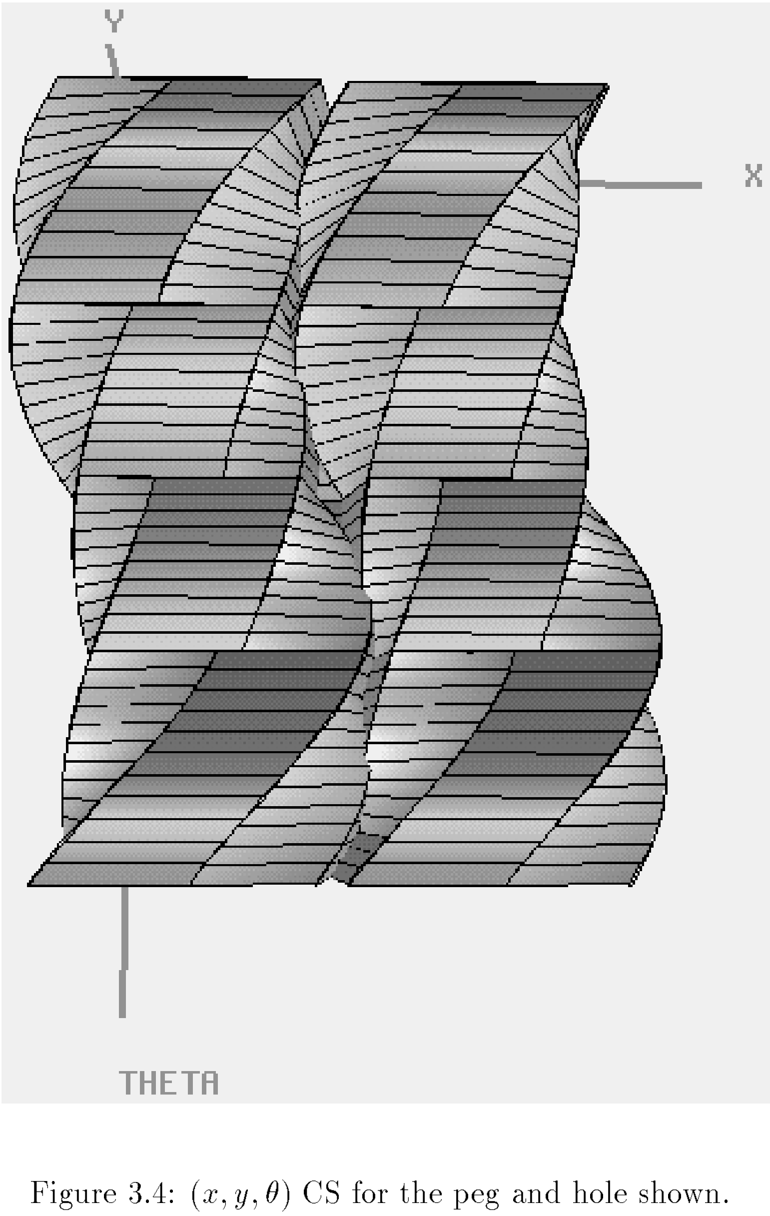


Figure 3.4: (x, y, θ) CS for the peg and hole shown.

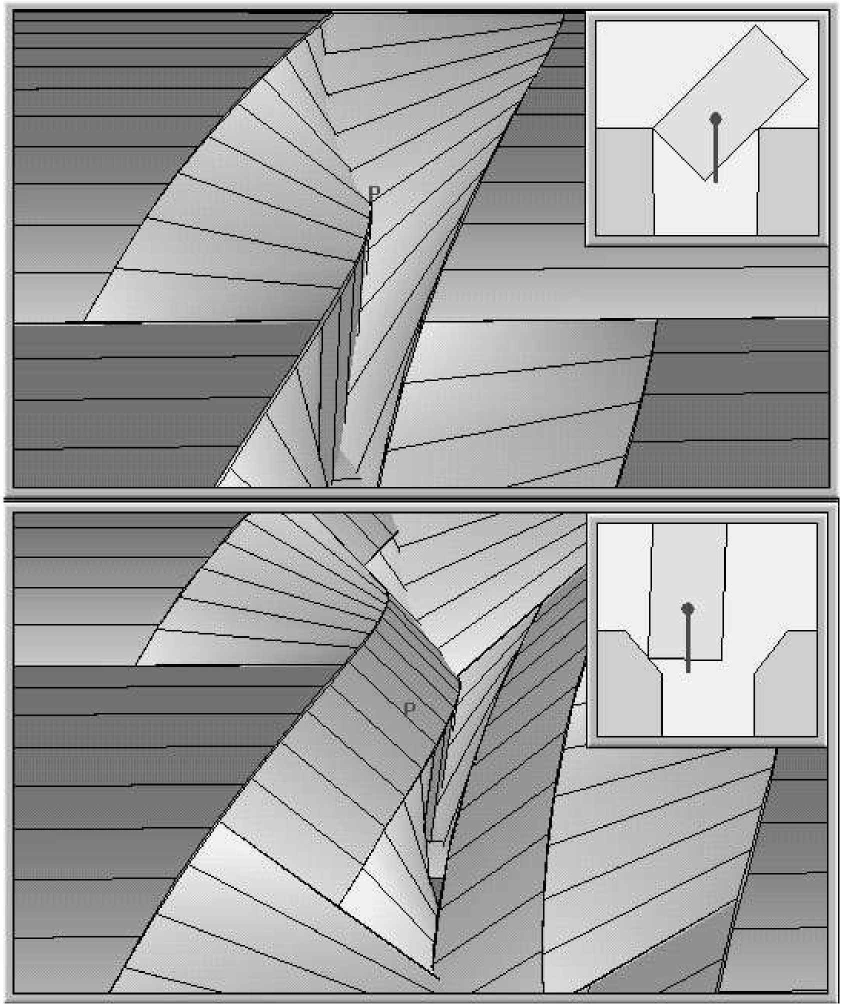


Figure 3.5: Entry regions on the CS for **(a)** a tilted peg and hole (intentional θ offset), and **(b)** a chamfered peg and hole.

center (reference point) on the peg, and non-kinematic parameters including the coefficient of friction μ and compliance matrix $[C]$. In Section 4.5 we will examine in greater detail the relationships between these parameters as well as develop the tools and strategies that will allow us to manipulate them for the purposes of designing more reliable assemblies.

3.2 Vibratory Bowl Feeders

In Section 1.1 we gave a very brief description of a portion of a vibratory bowl feeder track used to orient small parts. We will now complete this description and develop the appropriate representations to model feeder function. Figure 3.6 shows a typical vibratory bowl feeder used to sort small parts for an automated assembly system. A large number of unoriented parts are placed in the bowl and driven up the spiral track on the bowl's interior by vibratory motion. Parts reach the top of the track in single file in one of a finite number of stable orientations as shown in Figure 3.7. As the parts reach the top of the track, they pass through a series of features built into the track and bowl wall designed to *(i)* reorient certain part configurations, or *(ii)* reject parts in an undesirable orientation by causing them to fall off the track and back into the bowl to be recirculated. Some typical track features are shown in Figure 3.8. The desired result is a series of parts in a single known orientation exiting the outlet of the feeder to be placed into a fixture or pallet by a robot or other transfer device.

One could argue that the term **feeder** used to refer to the device consisting of the bowl and track is something of a misnomer in that the bowl and track geometries only perform the function of feeding for a specific part or set of parts. Parts other than those for which the bowl and track were designed would not be fed properly if placed in the bowl. Thus, from a functional standpoint, the "feeder" is distributed between both the part geometries and the bowl/track geometries with which the parts interact.

The second form of part *feeding* operation listed above (*(ii)* reject parts in undesirable orientations) can be characterized as a filter on part *motions*. In the vibratory bowl feeder, parts moving along the track in a number of different initial orientations interact with bowl and track geometries like those shown in Figure 3.8 and undergo different classes of motions depending on the characteristic constraints imposed by those interactions. We identify two broad classes of motions consisting of an *accept* motion in which part is allowed to continue to remain on the track, and a series of *reject* motions where the parts are forced off of the track by removing their support. Figure 3.9 illustrates a motion constraint metaphor for this process where a series of discrete motions, corresponding to parts traveling in each of the initial stable orientations, are filtered into either a single accept motion or a series of reject motions

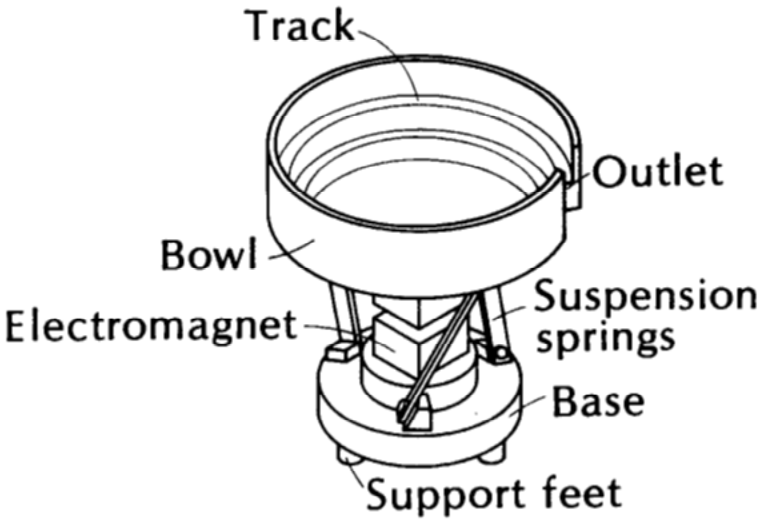


Figure 3.6: A vibratory bowl feeder. The bowl at the top is driven in a combined vertical and rotary oscillatory motion by the electromagnet and springs at the bottom, causing small parts placed in the bowl to move in single file up the spiral track on the bowl's interior. From Boothroyd et. al. [7].

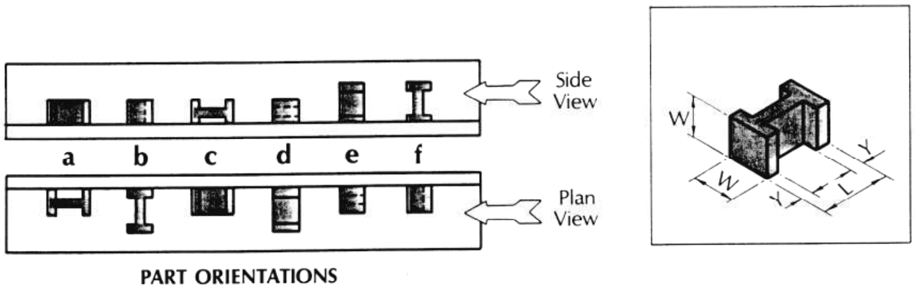


Figure 3.7: Parts arrive at the top of the track in one of a number of stable orientations. From Boothroyd et. al. [7].

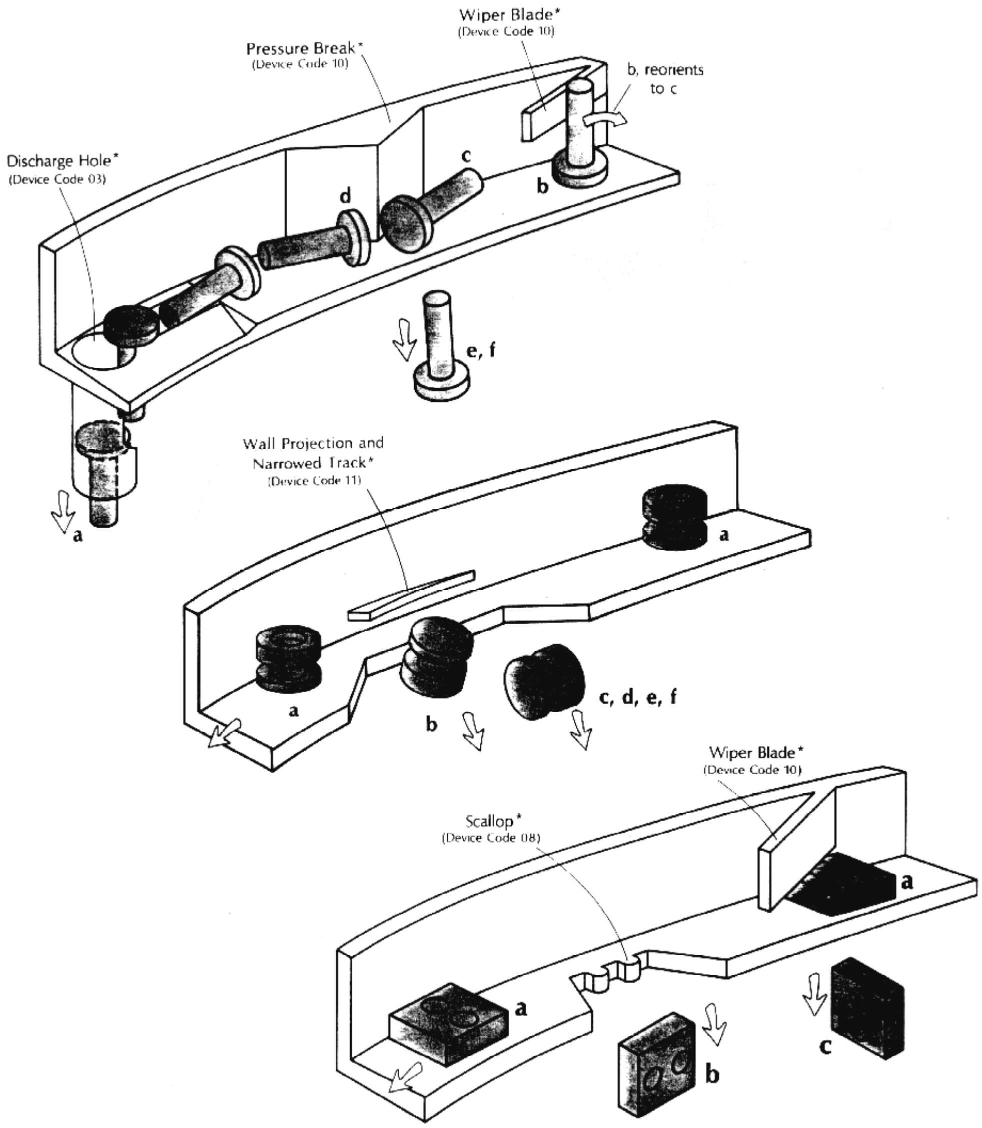


Figure 3.8: Some common bowl and track features designed to reorient or reject parts in certain orientations. From Boothroyd et. al. [7].

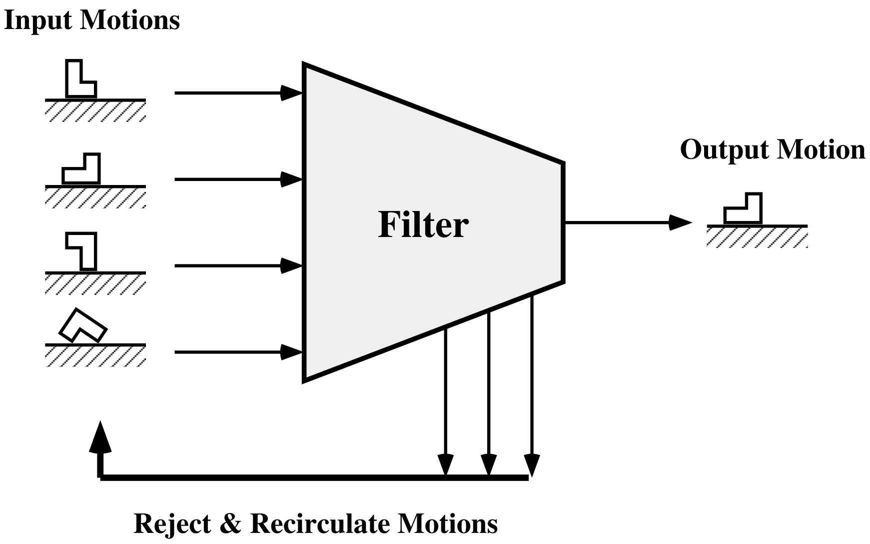


Figure 3.9: A motion filter metaphor for the function embodied in the part/feeder interactions.

that return the parts to the bowl for another round.

The interactions we will focus our attention on will occur within the region encompassing the part/bowl/track interactions near the outlet of the feeder.[4] To model the part/feeder interactions we will assume the parts to be traveling in the plane of the track as shown in the top of Figure 3.10, with both the bowl wall and track modeled as being (locally) straight. Since we will not explicitly model falling motions out of the plane, we will further simplify the model to the planar polygons by taking appropriate slices of the objects as shown in the bottom of Figure 3.10, where we view the track from above looking down along the negative z axis. The part, bowl wall and track are modeled as polygons, with the bowl wall polygon at the bottom of the figure and the track polygon underneath both the part and bowl wall polygons. Parts enter from the left of in the figure (bottom) and slide to the right while in contact with the bowl wall while being supported by the track. Parts that fall off the track return to the interior of the bowl located above the edge of the track ($+y$ direction).

The exact motion of a part along the track can be quite complex. The left half

[4]We should note that this set of motion constraints actually comprises the *second* stage of filtering part motions since parts arriving at the top of the spiral track have already been, in effect, pre-filtered into one of the stable orientations during their journey up the track. This first filtering operation depends solely on the geometry of the part itself since the feeder geometry up to this point consists of a simple wall.

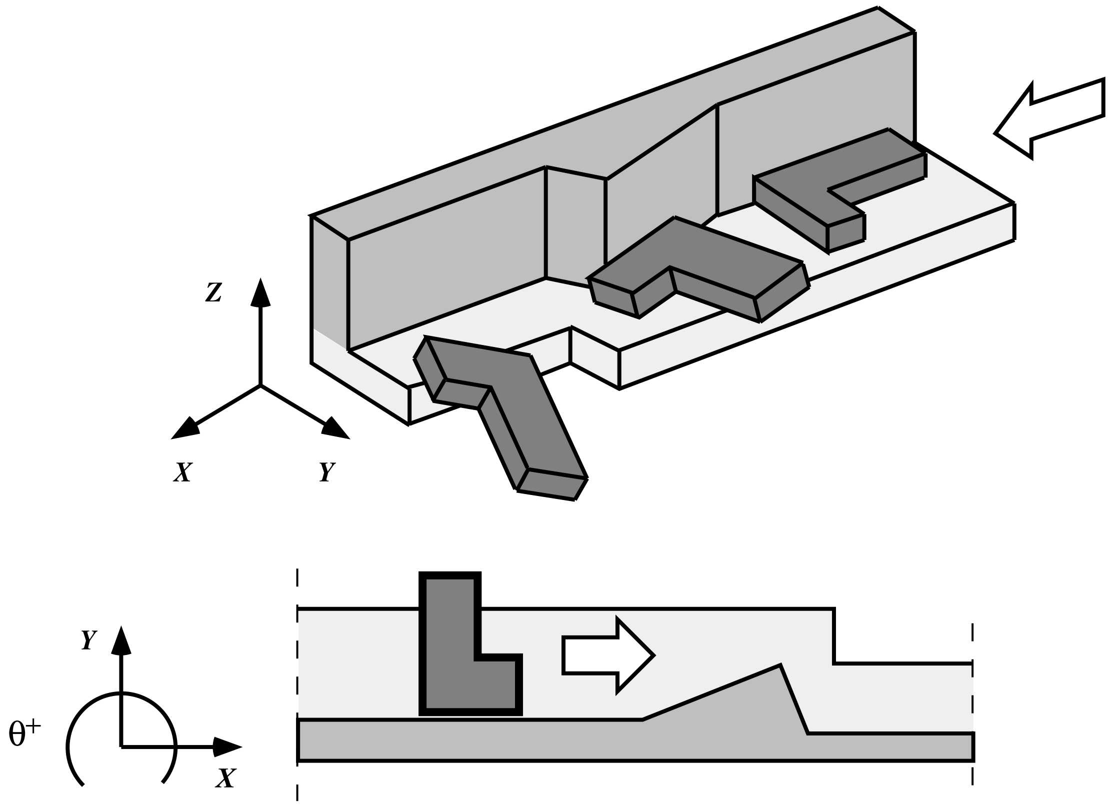


Figure 3.10: A polyhedral model of a portion of the feeder track near the outlet of the bowl (top), and an equivalent planar model viewed from above (bottom).

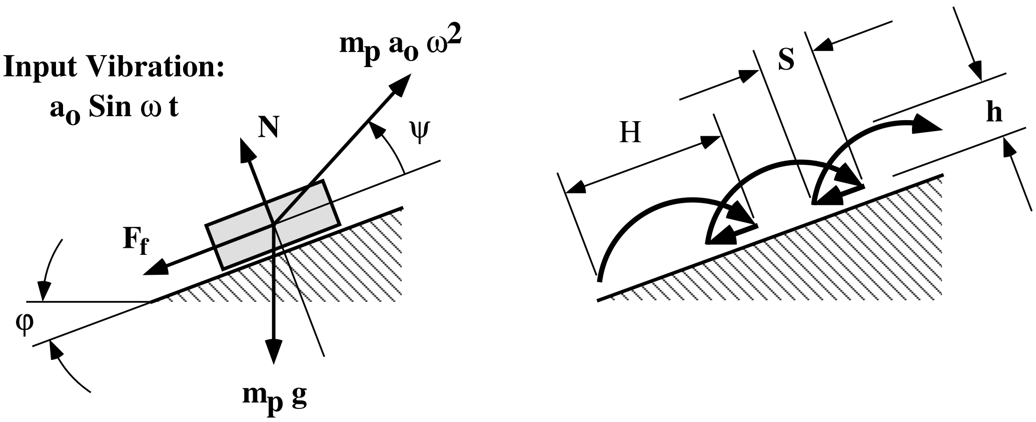


Figure 3.11: Force equilibrium for a part in contact with the track (left), and a typical hopping-sliding motion for a part on the track (right). Both illustrations are in the vertical plane containing the gravity vector g.

of Figure 3.11 illustrates the forces acting on a part in contact with the track in a vertical plane containing the gravity vector g where m_p is the mass of the part, ϕ is the inclination of the track relative to the horizontal, $a_0\omega^2$ is the acceleration of the bowl due to vibration, ψ is the angle between the vibration acceleration and the track surface, N is the normal and F the frictional forces between the part and the track. The right half of Figure 3.11 illustrates the various modes of part motion including hopping and sliding, where the hopping height h is normal to the track, and the hopping distance H and sliding distance S are both parallel to the track. For most reasonable values of the vibration parameters a_o, ω and ψ, the hopping height h is one to two orders of magnitude *smaller* than either of the sliding motions H and S.[5] To simplify the analysis, we will approximate the overall motion of the part on the track as being purely sliding. Furthermore, we will combine the part's mass and the applied accelerations due to vibration and gravity into one force applied to the part's center of gravity (i.e. CS reference point), and will only consider the components parallel to the (x, y) plane of motion. Finally, since the average velocity of the part's macroscopic motion is constant and relatively small, we will model the dynamics of part motion as being quasi-static.

Figure 3.12 shows the CS and support constraints in configuration space for a planar vibratory bowl feeder example (from Figure 1.3 **(a)**). The CS represents the kinematic motion constraints due to interactions between the part and bowl wall only. The highlighted regions on the surface of the CS represent those points

[5]Often feeder tracks are covered with a rubber coating to absorb impact energy, i.e. $e \ll 1$, in order to keep the part motions deterministic (see Boothroyd et. al. [7]).

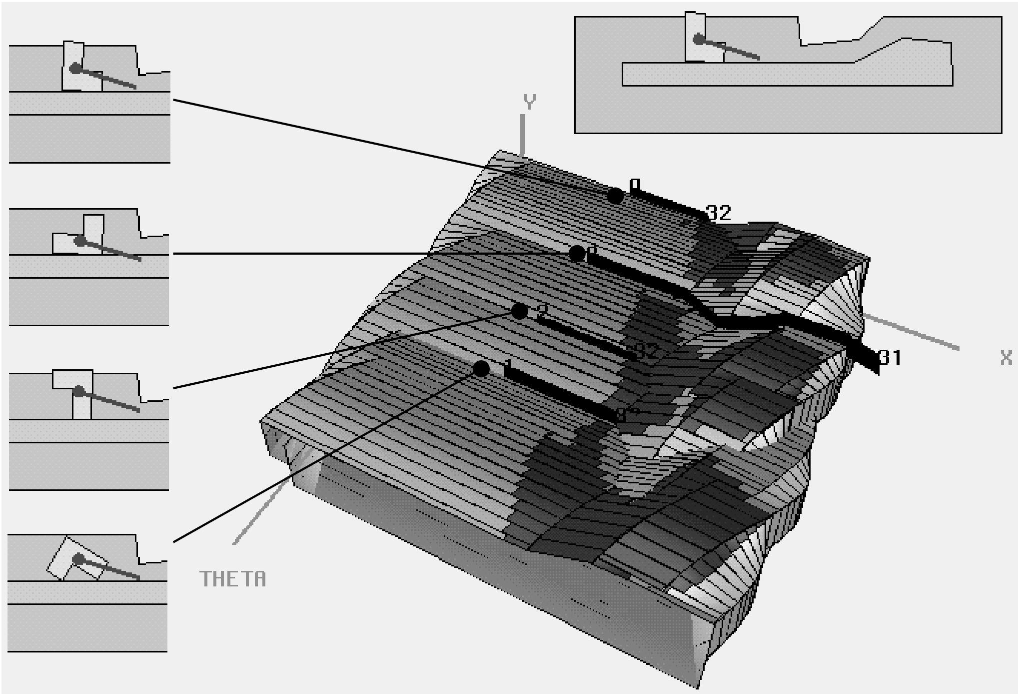


Figure 3.12: Configuration space representation of a planar bowl feeder example.

in configuration space where the part is in contact with the bowl wall but is **not** supported by the track (see Sections 2.4.3 and 2.5). The paths illustrated on the surface of the CS represent motions of parts along the track in contact with the bowl wall from each of the stable initial starting orientations (also shown). The thickness of each path is drawn proportional to the relative probability that a part will enter the feeder in that initial orientation, with the thickest paths having the highest probability and the thinnest having the least. In the example shown, we notice that all but one of the motions paths enter one of the unsupported regions and terminate at the boundary marking the (x, y, θ) position where the part will fall off the track and back into the bowl. The one remaining path exits the feeder to the right, corresponding to the outlet of the bowl feeder. Recalling the filter metaphor of Figure 3.9, the finite number of discrete paths entering the feeder are filtered into one **pass** motion (path number 3), and a number of **reject** motions (paths $0, 1, 2$). The result is an explicit representation of the feeding function derived from the interaction between the part and the feeder geometries.

We will examine the vibratory bowl feeder domain in greater detail in the next chapter on design. For the moment we will briefly discuss some of the major characteristics of feeder function as represented in configuration space. We begin by tracing the motion paths shown in Figure 3.12 as they move from the initial positions across the CS. One of the most noticeable features on the CS is the series of "valleys" parallel to the x axis and offset from one another in θ, each containing one of the numbered initial part orientations at the beginning of each path. As noted in Section 2.3.2, the CS edges forming the bottom of these valleys occur along type B facet adjacencies where an edge of the moving polygon (part) and the stationary polygon (bowl wall) are in contact, leaving one remaining degree of freedom for the part. These valleys, and the type B facets that bound them, form a pre-filter which divide parts placed into the bowl in purely random orientations into the set of discrete stable orientations shown. The motions along the one-dimensional lines at the bottom of these valleys are extremely stable and relatively insensitive to changes in dynamics parameters. By the same token, the θ positions of these valleys and the curvature of the facets forming them are determined by the geometry of the part as it interacts with the straight bowl wall. As a result, little further differentiation between motions of the part in different orientations is possible.

The next set of CS features encountered by the motion paths is a "ridge" running roughly parallel to the θ axis. This ridge, composed of both type A and type B facets, serves to move the parts closer to the track edge ($+y$ direction) as well as disperse the motion paths from their fixed-θ valleys. As the paths leave their valleys and cross these facets their degrees of freedom increase from one to two, making the motion paths more *susceptible* to (or *controllable* by) changes in the dynamics parameters. Finally, all but one of the paths encounter the edge of the track by entering unsupported regions of the CS where the planar path is terminated when the part falls off the track and back into the bowl.

The parameters that contribute to the motion constraints forming the CS are the geometrical features of the part and bowl wall. The part and track geometries interact to determine the supported regions of configuration space, which are then intersected with and represented on the CS surface. Finally, the dynamics parameters represented by an applied force representing the combination of gravitational and vibrational accelerations, and the coefficient of friction round out the list of parameters that go in to creating the full set of motion constraints shown in Figure 3.12.

3.3 Fixtures and Pallets

Figure 3.13 shows a typical fixture/pallet used to locate and hold parts for transport, light machining and assembly. This class of fixture falls into the class of static, or unarticulated, fixtures (as opposed to those that contain manual clamps or are otherwise actuated). The function of a fixture is to secure a part or subassembly in a known position and orientation, and to keep it in that configuration in the presence of whatever forces may be generated during the above operations. Characteristics

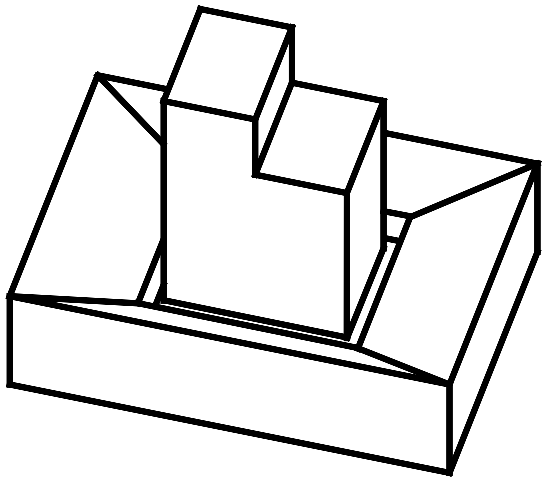


Figure 3.13: An illustration of a typical fixture/pallet used in automated assembly.

that are desirable for a fixture include:

1. the ability to hold parts in a known configuration that is also easily and reliably reachable when placing the part in the fixture,

2. the ability to hold parts stably under a variety of external loads applied to the part,

3. provides access to the parts by other parts, grippers, or tools as necessary, and

4. the part or subassembly may be removed once the desired operation has been performed.

There are a number of other fixture characteristics not listed above whose relative importance depends on the application. By no means the least of these other characteristics is the requirement that the fixture be as inexpensive and quickly producible as possible. Fixtures are a component of virtually every manufacturing system, both manual and automated. As noted in the section on vibratory bowl feeders, a considerable portion of the fixed capital cost for a manufacturing system consists of feeders and fixtures. Since the geometry of the fixture depends heavily on the geometry of the parts and the type of manufacturing operation, fixtures must typically be custom designed for each application. Therefore, tools and techniques that would improve the productivity and performance of fixture design would be extremely valuable.

Functionally, a fixture shares a great deal in common with an assembly. The goal, in terms of motions, is to place a part into the fixture in a known position and

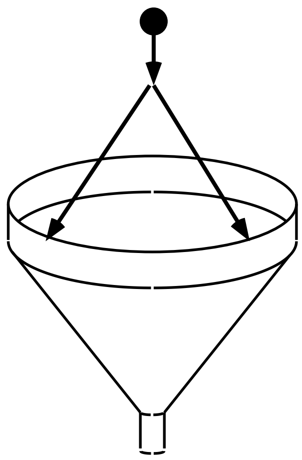


Figure 3.14: Functional metaphor for dropping a part into a fixture in terms of motion constraints.

orientation in the presence of uncertainty. Like Figure 3.3, the kinematic motion constraints due to the part/fixture interaction may be considered to form a funnel which, in the ideal case, will guide the part to the desired location from a range of initial positions. The main difference between fixtures and assembly is the nature of the forward projection constraining the motions of the part. Whereas we considered an assembly to consist of a compliantly held part (peg) guided along a nominal trajectory by a device such as a robot, we view the task of inserting a part into a fixture as one of dropping the part into place from some height in a gravity field. As a result, the bounded-energy forward projection for conservative systems, as shown in Figure 3.14, is a more appropriate model. [6]

As for assembly, we will use a planar model of a part and fixture, with gravity assumed to lie in the plane of the figure pointing down, as shown in Figure 3.15. Like the assembly model, the plane in which (x, y, θ) motions may take place is chosen to capture the relevant geometrical features of both the part and the fixture. For axisymmetric objects the plane is chosen to contain the axis of symmetry. We assume that the part is dropped with zero initial velocity above the fixture from within a bounded range of initial positions and orientations. The coefficient of friction μ and coefficient of restitution e for the two parts are assumed to be known constants.

[6] We consider those cases where grasped parts are placed into the fixture using compliant motion to be an assembly, and model them in the same way as described for the peg in hole example. In the context of motion constraints, there is no distinction between a fixture and a subassembly in such cases.

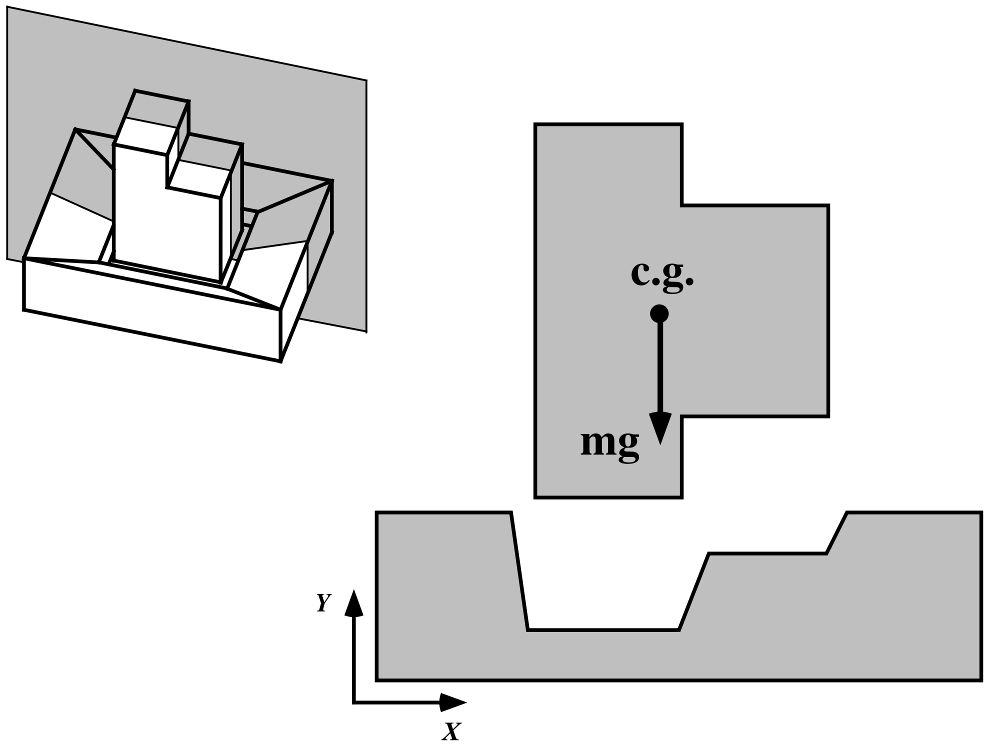


Figure 3.15: A planar model of a part dropped into a fixture.

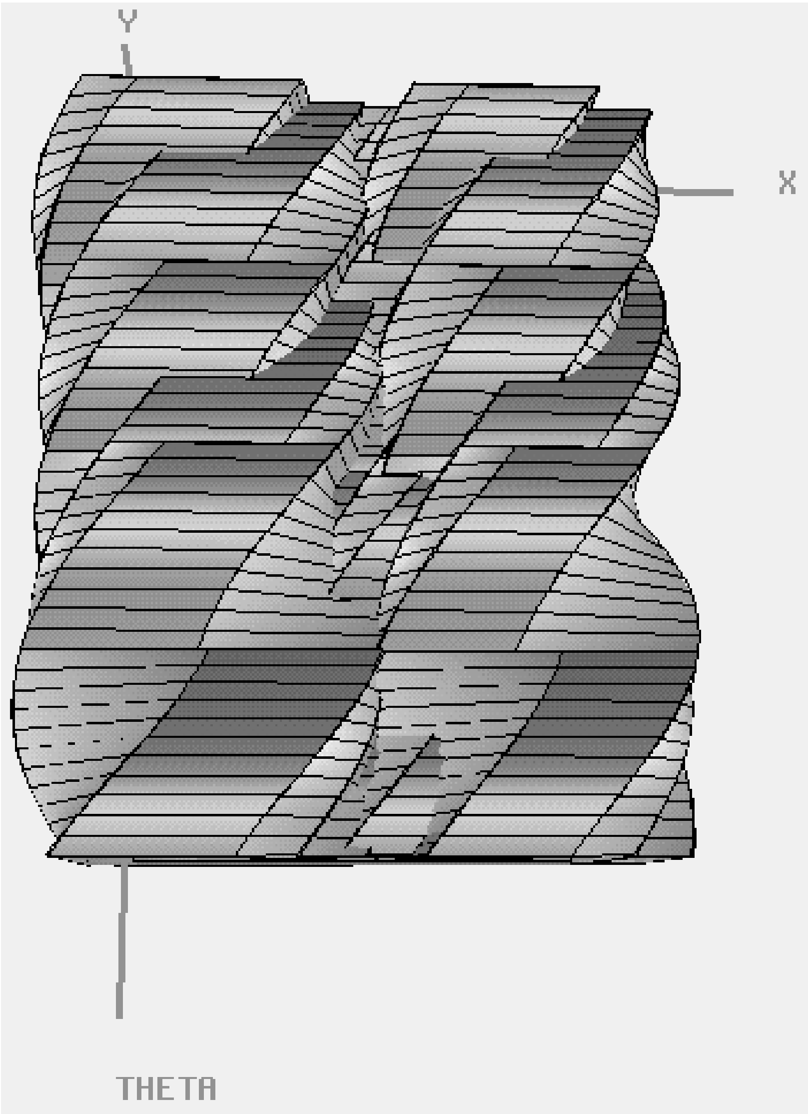


Figure 3.16: Configuration space representation of a planar part and fixture example.

The configuration space representation of the part/fixture interaction is shown in Figure 3.16. We see a strong resemblance between Figure 3.16 and Figure 3.4 showing the CS for the peg in hole example. The "throat" of the CS hole for a fixture is somewhat wider and shallower in comparison to that for the peg in hole assembly, but the entry region around the hole has many of the same funnel-like characteristics intended to guide motions toward the goal state at the bottom of the hole.

The primary difference between the fixture and assembly is the use of an energy bounded forward projection model rather than that of a path produced by compliant motion (see Section 2.4.2). The resulting forward projection would appear as a polyhedral cone, similar to that illustrated in the lower half of Figure 2.8, with its central axis parallel to the gravity vector in the (x, y, θ) configuration space.[7]

The CS and forward projection representations described above address the first of the desirable characteristics for a fixture listed above, namely the ability to get a part into a known configuration and keeping it there. To address the second issue of holding a part stably under a variety of loads once the part has reached the goal configuration, we return to our discussion of the configuration space friction cone. We recall from Section 2.4.1 that the friction cone spans the set of reaction forces that will maintain equilibrium for an object. The negation of this cone, therefore, represents the set of applied forces, and torques, that may be *applied* to the part without any motion resulting. The larger the span of the friction cone, the more stably the part will be held in the presence of applied loads. Of course, the caveat is that the same frictional effects that help keep the part in place are also the frictional effects that can make getting the part into the desired configuration more difficult.

The third desirable characteristic of a fixture providing access to the secured part by other parts, grippers, or tools has to do with the kinematic constraints between the fixture and these other objects. Specifically, we wish to determine if the gripper/part/tool will come into contact with the fixture during it's operation on the part being held in the fixture.[8] In terms of the above representation, if the reference point of the moving object is chosen to coincide with the reference point (i.e. **cg**) of the part when that object was interacting with the part (grasping it, for example), then the CS for the object/fixture interaction could be *superimposed* onto the part/fixture CS to produce the complete set of kinematic motion constraints on the part, fixture and object. This superposition is the same as that discussed in Section 2.5 for superimposing CS constraints for interacting polygons representing multiple slices of three dimensional objects.

[7]For simplicity, our example fixture is assumed to be frictionless. For non-zero friction we would also need to add representations for sticking regions on the surface of the CS, similar to those generated by Brost [13], where a part might become stuck.

[8]This is precisely the collision avoidance problem to which the configuration space representation was initially applied to robotics for planning purposes.

Finally, the fourth characteristic desirable in a fixture, that of being able to *remove* a part from the fixture, is addressed both by the representation that determines if the part may reach the goal after being dropped, and the ability of another object such as a gripper to reach the part in the fixture in order to remove it – the basic assumption being that if you can get it in and can still grab it, then you can get it out again.

As with the peg in hole example, the ease and reliability with which a part may be placed in the fixture is determined by the motion constraint facets forming the entry region of the CS hole, or in this case the subset of those facets that lie within the forward projection of the dropping motion. To improve the reliability of this motion we may:

- vary the fixture/part geometry that defines the shape of the constraint facets to improve the funnel characteristics of the entry region similar to assembly, as well as changing the size and location of the sticking regions on the CS,

- vary the coefficient of restitution e that determines the span of the forward projection cone by changing the materials used for the fixture, and

- vary the coefficient of friction μ that determines the extent of the sticking regions on the surface of the CS, also by changing the materials used for the fixture.

Specifically, for the dropping task, we are concerned with the region of configuration space bounded above by the forward projection cone and below by the kinematic motion constraints of the CS. The funnel-like entry region on the CS is determined by the part and fixture geometries, whereas the forward projection cone is determined by the initial set of dropping positions and e. We note that here, as in many of the other examples, shape is generally the easiest to change of the parameters that may be modified since the choice of fixture materials may be limited by other factors. For changing the stability of the fixture, the span of the friction cone at the goal configuration is a function of both the CS (i.e. part and fixture geometry), and μ. And finally, the accessibility and removability of the part in the fixture as represented by the superposed CSs for the part/fixture and gripper/fixture are functions of the geometries of those three objects.

We have included the part-fixture example to round out the set of examples in which function may be represented and visualized in terms of motion constraints. Much work remains to be done in implementing detailed energy-bounded forward projection models for both the conservative and non-conservative cases described earlier in this chapter. Therefore, we will not consider this example in any more detail within this report. The reader is referred to extensive work on the analysis of planar dropping tasks found in Brost [13].

3.4 APOS

Another form of vibratory feeder developed more recently for orienting small parts is the **A**dvanced **P**arts **O**rienting **S**ystem (**APOS**) developed by the Sony Corporation. APOS, shown in Figure 3.17, consists of a vibrating pallet into which has been machined a series of cavities, or wells, designed to capture parts in a known orientation. A cluster of hoppers containing different sets of randomly oriented parts is located near the top of the pallet which is angled slightly down and away from the buckets. When a small gate in one of the hoppers is opened a cluster of parts falls onto the vibrating pallet where the parts hop and slide downhill across the surface of the pallet. A part that happens to be near the desired orientation will fall into one of the empty cavities and be held there. Parts that are not captured by one of the cavities continue down the pallet where they fall into a return bucket. The return bucket is then periodically lifted and its contents dumped back into the hopper. This cycle is repeated for a predetermined period of time so that a majority of the cavities will contain an oriented part. At the end of the cycle the vibration is stopped, and excess parts are cleared from the pallet by an air jet, and the pallet is transferred to a conveyor and carried off to a robotic assembly station. Figure 3.18 illustrates the major operations of APOS.

In economic terms APOS has a number of advantages over the more common vibratory bowl feeder. First of all, most of the hardware components in APOS are reusable for new products and production lines – the only hardware component that must be custom designed for a given part geometry is the pallet. Another advantage of APOS is that it combines into one unit a number of assembly system components that are typically separate in other systems, including: feeder, fixtures, and pallets for parts transport.[9] APOS thus combines into one compact unit a number of typically distinct manufacturing subsystems, most of which are reusable.

The one aspect of APOS that proves to be the most difficult and time consuming to develop is the pallet geometry containing the shaped cavities.[10] Figure 3.19 illustrates a number of "generic" pallet geometries designed to handle different classes of parts [46]. The pallet geometries shown are used to initially orient parts into one of a limited number of orientations. As the parts move down the slots and channels they come across, and some are trapped by, cavities machined into the pallet. In the figure, parts are dropped from the hopper onto a pallet at the top right and hop and slide down the pallet surface toward the lower left. A pallet of the type shown in **(a)** is used to capture flat symmetrical parts, like gears, whose orientation in the plane is not critical. A pallet of type **(b)** is typically used to orient long thin parts such

[9]Some recent work has even considered APOS for use in parts assembly directly. See Moncevicz [58].

[10]In addition to individual pallet designs, different parts often require individual vibration profiles which are stored in a programmable controller.

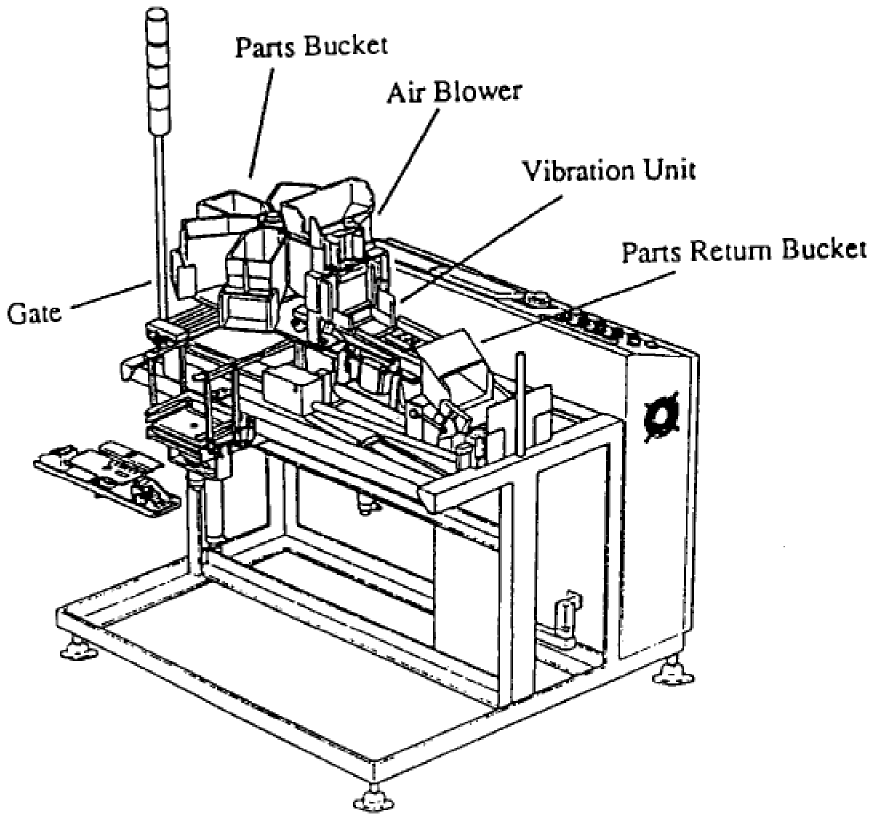


Figure 3.17: The APOS vibratory parts feeder developed by Sony. The vibrating parts pallet is in the center of the machine. From Fujimori [32].

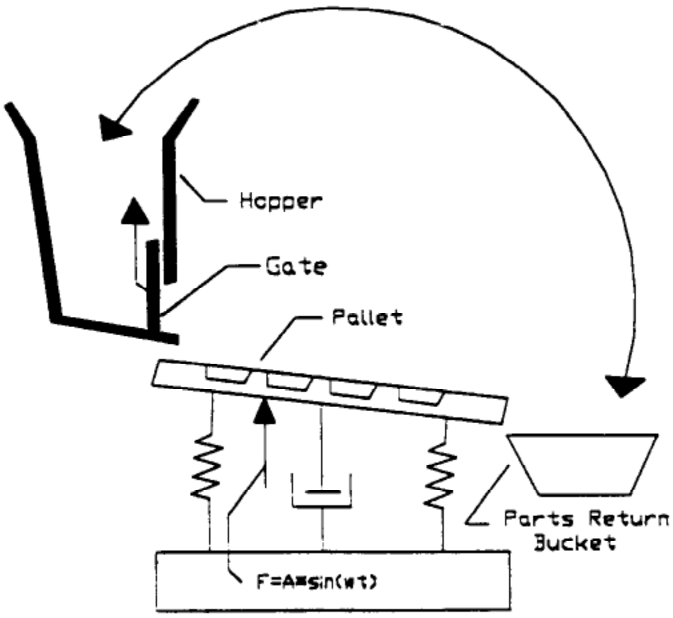


Figure 3.18: A side-view schematic of the APOS feeder, from Moncevicz [58].

as screws and springs that slide into the slots in a preferred orientation. A pallet like **(c)** is the most common type and is typically used for less symmetrical parts which fall into the "saw-tooth" valley and against the vertical wall in one of a limited number of orientations, similar to a bowl feeder track. And finally, a pallet of type **(d)** is used to orient parts that, after falling into one of the cavities, might block the flow of following parts. It is expected that parts in the correct orientation will fall into a cavity and remain there whereas parts in the wrong orientation may fall partially into a cavity, but should eventually bounce back out again. Characteristics that are desirable for an APOS pallet include:

1. the ability to trap only those parts that are in a desired configuration,

2. the ability to hold trapped parts stably after the feeding operation, i.e. after the vibration has been stopped, while the pallet is unloaded and transferred to the assembly station, and

3. the pallet provides access to parts by a robot gripper so the part may be reliably grasped and removed from the pallet for assembly.

Comparison of these characteristics with the vibratory bowl feeder and fixture examples suggests a considerable degree of functional overlap with APOS.

Like the vibratory bowl feeder, the function of APOS can be viewed in terms of a filter on part motions. Parts moving along the pallet surface interact with pallet features like those shown in Figure 3.19 and undergo different classes of motions depending on the characteristic constraints imposed by those interactions. As before, we identify two motion classes: *accept* and *reject*. In the APOS example, however, the accept motion consists of a motion termination, or trapping motion, in which the part is stopped and held in the desired configuration, while the reject motion for a part consists of all other motions where the part continues across the pallet surface and into the return basket. Interestingly, the characteristics of the accept and reject motion classes for APOS are the reverse of those for the vibratory bowl feeder. Figure 3.20 illustrates a motion constraint metaphor for APOS, which is identical to that of the bowl feeder except that the multiple *output motions* are the reject motions, and the single accept motion is, in essence, a null motion corresponding to the part being contained within the vicinity of a goal region into which it will settle when the vibratory motion is stopped.

Depending on the type of part and pallet used, the hopping motions of the 3D parts on an APOS pallet are not always planar in a general sense. Parts dropped onto a pallet such as Figure 3.19 **(b)** are constrained to move within narrow slots so that the resulting motions all occur in more or less a vertical plane. Flat parts such as gears, plates, brackets, levers, etc. that are dropped onto pallets such as Figure 3.19 **(a)**, **(c)** or **(d)**, move more or less in the plane of the pallet and can often be viewed as planar sliding motions with minimal vertical motion, similar to vibratory

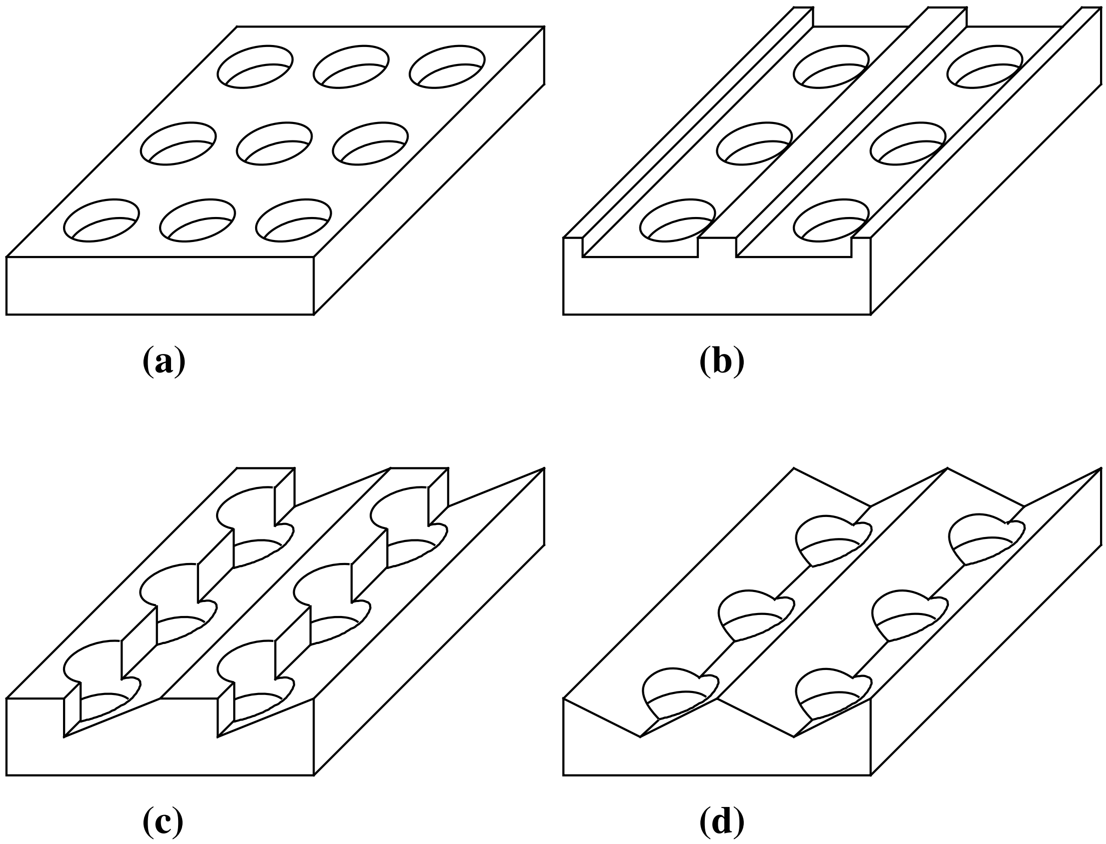


Figure 3.19: A taxonomy of four "generic" APOS feeding pallets.

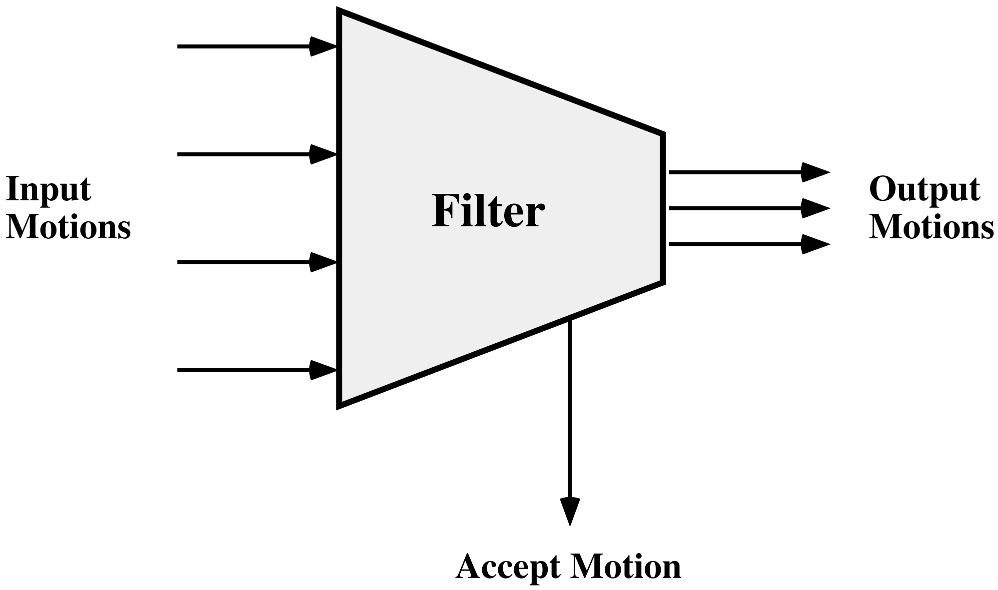


Figure 3.20: A motion filter metaphor for the function embodied in the part/feeder interactions within APOS.

bowl feeding. Other more complicated parts will typically find themselves in one of a number of stable orientations with motions occurring in primarily horizontal or vertical planes (possibly both), which can be used to capture the major characteristics of the part/pallet interaction. The applied vibration has horizontal and vertical components that are separately programmable, and in all of the above cases the amplitude of the vibration used is typically moderated by the desire to make gross part motions more or less deterministic, as in the case for vibratory bowl feeders.

For the purposes of this discussion we will consider the subset of part/pallet interactions whose motions may be characterized within a vertical plane, as illustrated in Figure 3.21. As in the fixture example, the coefficient of friction μ and coefficient of restitution e for the part and pallet are assumed to be known constants. The pallet oscillates within the plane of the figure, although the amplitude is considered negligible in comparison to the scale of the parts. The gravity vector is also in the plane of motion as shown.

Figure 3.22 shows the CS for a planar APOS example, along with an approximate representation of the non-conservative energy-bounded forward projection that has been projected onto the surface of the CS. By approximate we mean that the forward projection representation shown in Figure 3.22 was constructed using a modification of the support region implementation discussed in Section 5.5.2. It is presented here for illustration purposes only and is not the result of any actual computation

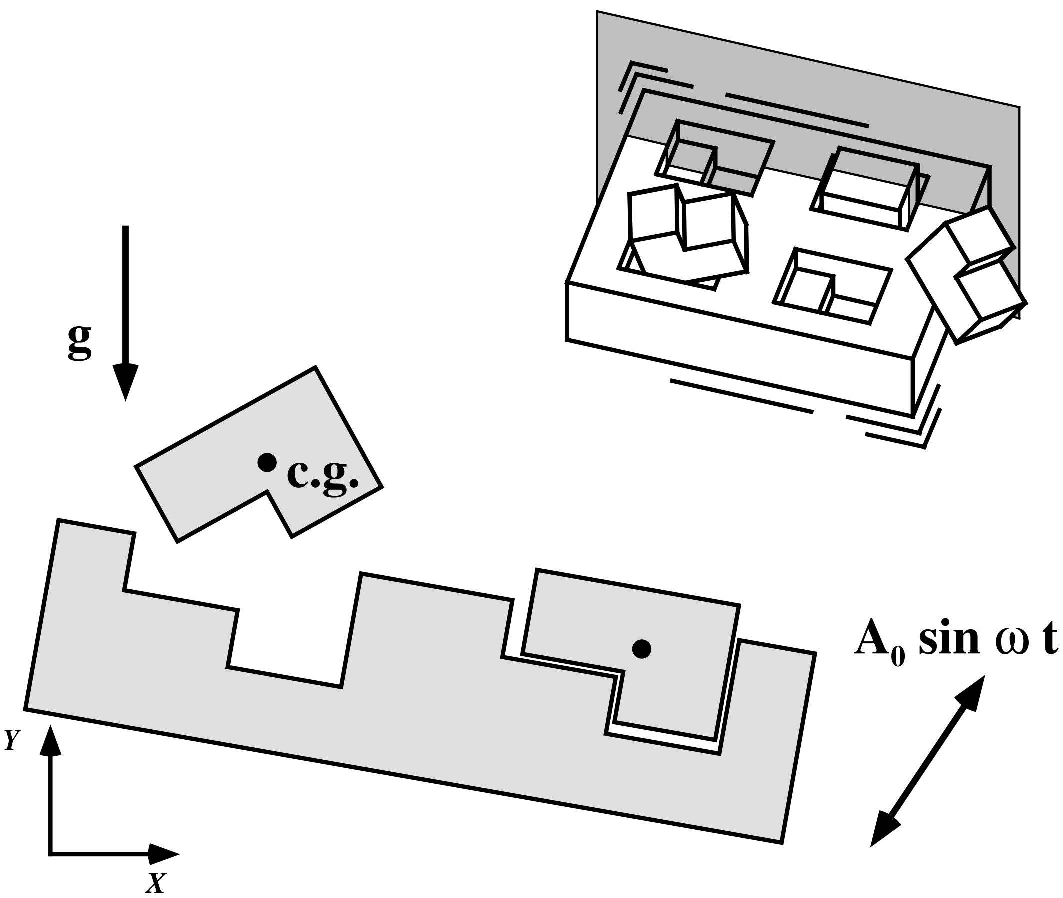


Figure 3.21: A planar model of a part and a vibrating APOS pallet.

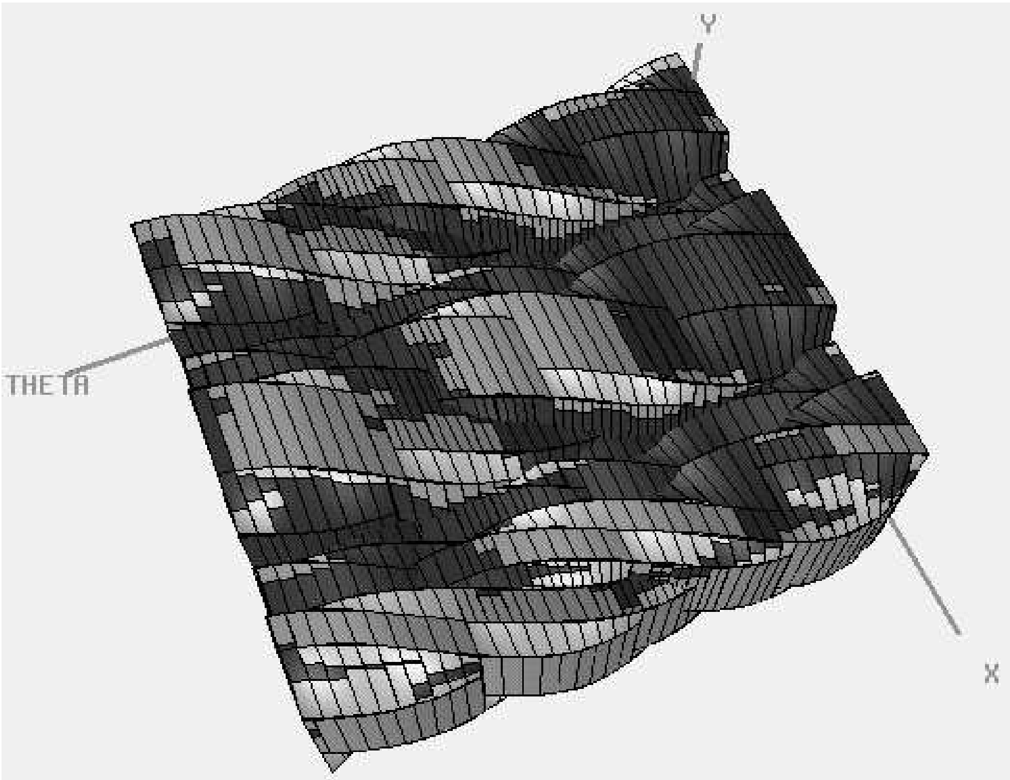


Figure 3.22: Configuration space representation of a planar APOS example. (Note: the forward projection shown is for illustration purposes only and is not the result of any actual dynamics computations.)

of bouncing dynamics. Conceptually, the highlighted regions on the surface of the CS represent the set of (x, y, θ) points in which the part and pallet may come into contact. The boundaries of these regions represent the intersection between the non-conservative bouncing forward projection motion constraints discussed in Section 2.4.2 and kinematic constraints of the CS. What is lost in this representation is any information about the positions in configuration space that may be traversed by a part in free flight, i.e. not in contact with the pallet surface.

For simplicity we assume that, as in the bowl feeder example, parts start out in one of their stable resting aspects on the flat portion of the pallet to the left in Figure 3.21. These points lie at the bottom of CS valleys of the kind found on the bowl feeder CS. The forward projection constraint boundaries extend outward from these starting points under the action of the applied vibration and gravity to envelope all reachable points on the CS surface. The result of this expansion is a "flow" of reachable points that spread down the CS valleys.[11] When the forward projection regions, or "rivers" if we may further extend the metaphor, reach the set of wells on the CS corresponding to the kinematic constraints between the pallet cavities and parts in each of the initial orientations, then the determination of whether or not a part is trapped in a cavity in a given orientation will depend on whether the river is able to overflow the well and continue to the right across the CS. For the part, pallet, and level of applied vibration to perform properly together as a feeder will require that only *one* of the river flows be stopped by a CS well while the rest overflow and continue off the right of the pallet. In terms of motions, this means that a part starting in one orientation will be caught and held in the cavity, whereas parts starting in other orientations will *eventually* bounce out of the cavity and continue across the pallet.[12]

We should note that the above requirements on the nature and extent of the forward projections on the CS for a successful APOS design are very conservative. For example, it is possible that if the forward projections overflowed all of the cavities in each valley of the CS that parts would still be captured in some of those orientations. The forward projection overflowing a cavity simply means that it is *possible* for a part in that orientation to leave the cavity. A more detailed model might contain embedded shells of forward projections, each with an associated probability that a part may reach the set of contacts contained within that shell. Of course, such a model would be considerably more complex than the existing (unimplemented) forward projection model. As noted earlier, much work remains to be done in implementing non-conservative energy-bounded forward projection model described earlier in this chapter.

In terms of APOS design, we wish to deepen the well surrounding the desired configuration *relative* to wells for other configurations so that the forward projection will be trapped only by that well. The CS facets forming each of these wells are determined by interactions between different part features, and the same pallet cavity features. The resulting CS facets, therefore, tend to exhibit a strong degree of coupling. As a result, modifying a pallet feature to deepen one well will often tend to deepen the surrounding wells. Careful attention to this coupling, as well as a considerable amount of trial and error, is required to arrive at pallet geometries that achieve the desired results.

[11] By way of analogy with the vibratory bowl feeder, we can imagine collapsing these energy-bounded "rivers" down to 1D space curves, at which point we would expect to see a motion representation similar to that of bowl feeders. An important point to note here is that, as discussed in Section 2.4.2, the exact motions of the parts on the APOS pallet will be *chaotic* in nature, and therefore impossible to compute exactly. The energy-bounded forward projection represents about the best we can expect to do in terms of predicting the behavior of parts in this system.

[12] We are neglecting the effect of a part trapped in a cavity on subsequent parts moving across the pallet. To explicitly capture these effects, it might be necessary to superimpose motion constraints generated using a modified pallet geometry consisting of the pallet and a trapped part.

In addition to kinematic constraints above, the forward projections on the CS surface will be determined by the dynamics parameters, including material properties of the part and pallet such as μ and e, as well as the orientation of the pallet relative to gravity ϕ, and the orientation, amplitude and frequency of the applied vibration ψ, A_0 and ω respectively.

As we have seen, APOS shares many of the motion constraint characteristics found in the other examples discussed so far, such as the presence of valleys on the CS corresponding to stable part orientations as found in bowl feeders, and constraint wells to capture parts like those found in assemblies and fixtures. The task of designing an APOS system is complicated by the heavy amount of coupling present between these motion constraints, which results from the fact that APOS itself combines so many functions into a single system. We should stress that this coupling is a reflection of the nature of the APOS system itself and **not** the configuration space representation. The representations discussed in this chapter are useful for this application precisely because they make this inherent coupling explicit. At present the computation of forward projection regions for actively driven vibratory systems, such as APOS, remain an open issue for further research.

3.5 Summary

Figure 3.23 shows the planar representations of the four example domains discussed in this section, and Figure 3.24 shows their corresponding representations in configuration space. We have noted a number of similarities between these representations throughout our discussion that are worth recalling here. First, we have focused a considerable amount of attention on the role of kinematic motion constraints represented by the surface of the CS and determined by interactions between object shapes. In the examples of assembly, parts fixtures, and the APOS feeder we saw how some of these constraints took the form of features on the CS surface that we likened to *funnels* or *wells* that guide motions toward a specific state or set of states in configuration space. In the bowl feeder and APOS examples we saw parallel *valleys* in the θ-dimension on the CS that acted to sort and guide parts into different stable orientations. In addition to kinematic constraints we saw representations of dynamic motion constraints in terms of forward projections determined by the mechanics of object interaction. For the peg-in-hole assembly and vibratory bowl feeder examples we were able to generate detailed representations of object motions as a set of paths, or trajectories, from initial states in configuration space and constrained by contact with the CS. For the fixture and APOS feeder examples we were unable to generate exact motion descriptions, but instead bounded the set of reachable states in configuration space through which any trajectory would pass.

In addition to their similarities, the four example domains were also chosen for

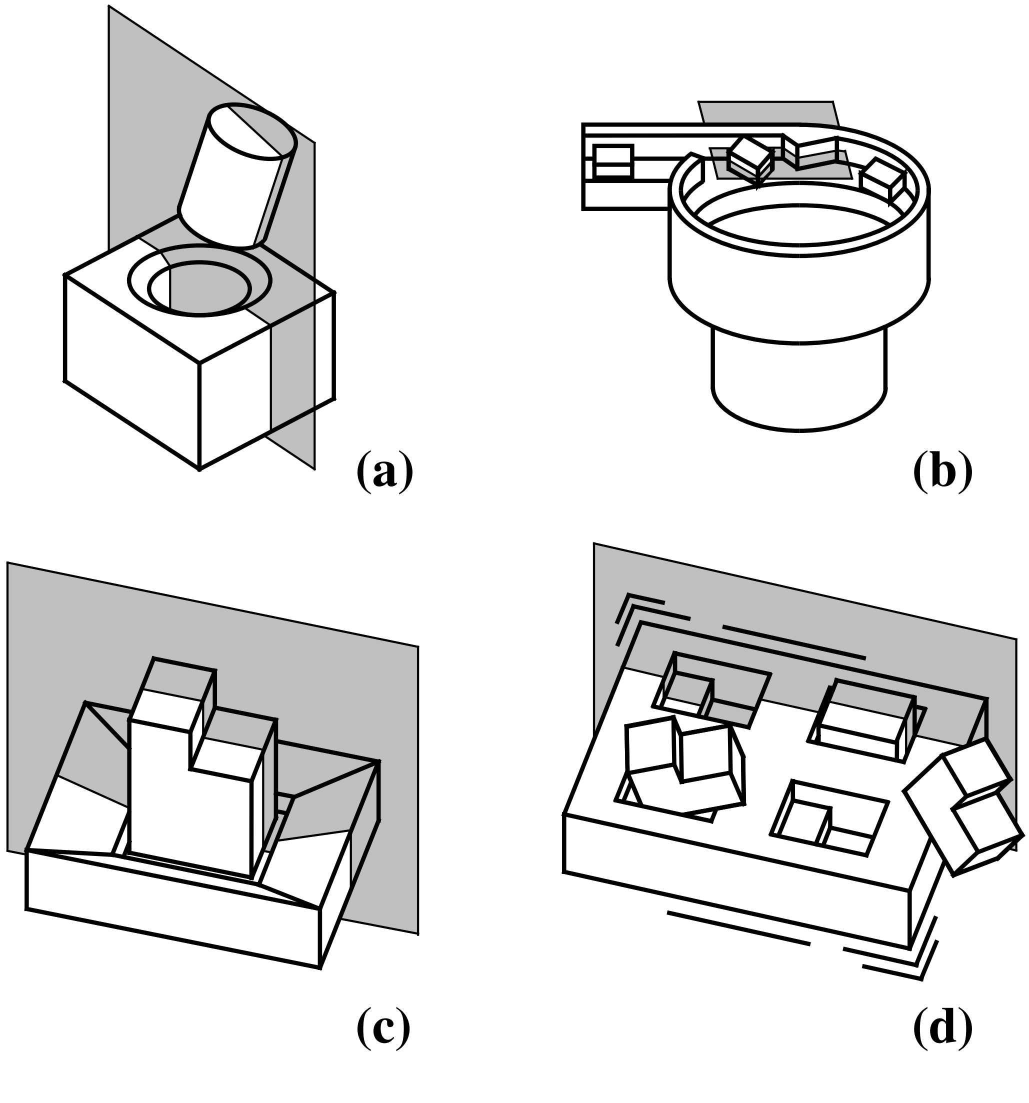


Figure 3.23: The four planar application examples: (**a**) peg-in-hole assembly, (**b**) vibratory bowl feeder, (**c**) fixture, and (**d**) APOS feeder.

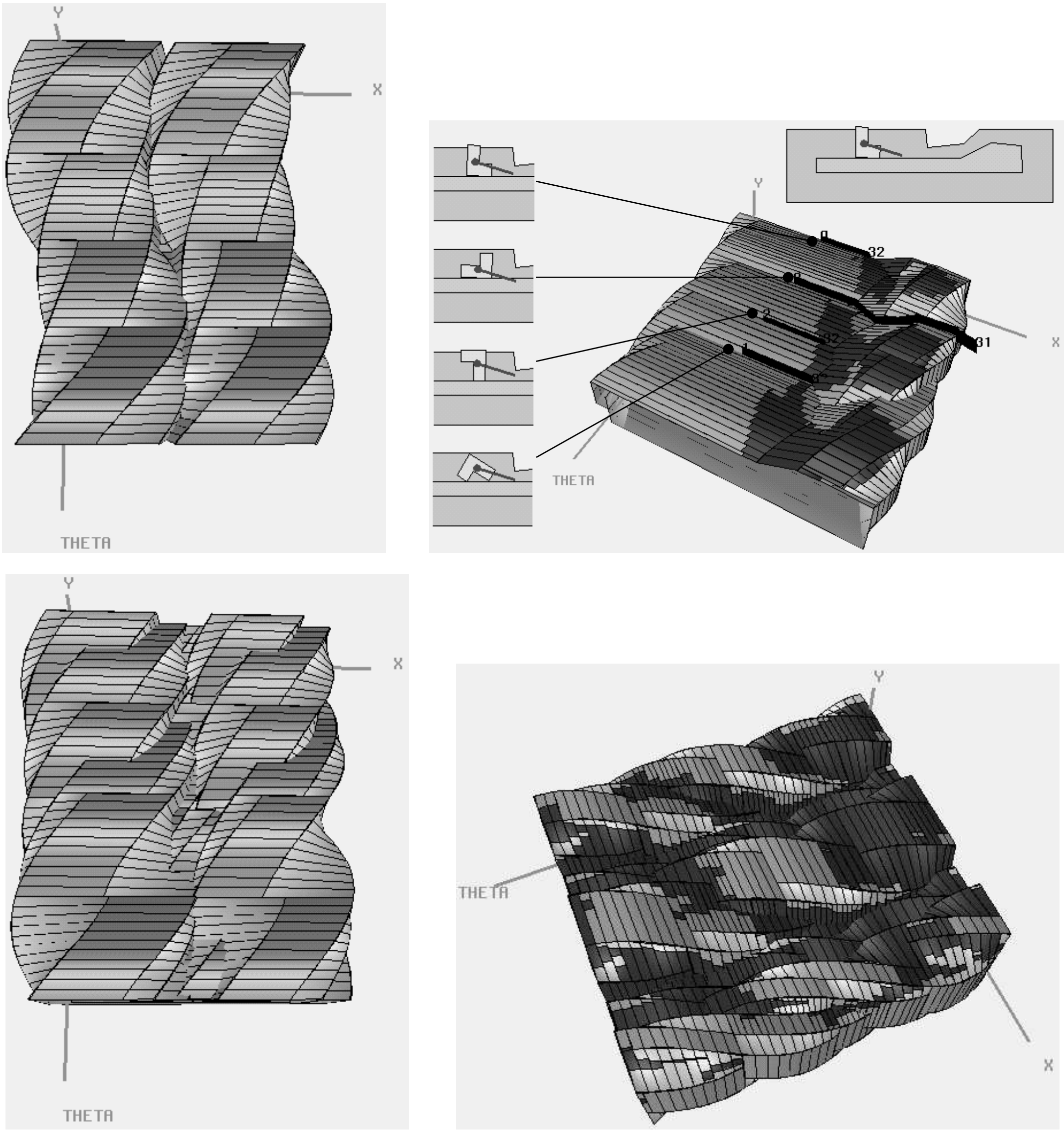


Figure 3.24: Configuration space representations for the four planar application examples: peg-in-hole assembly, vibratory bowl feeder, fixture, and APOS feeder.

their different usage of the resources made available within the configuration space representation. Specifically, both the assembly and fixture examples focused on a relatively small region of configuration space where the local set of CS facets were sufficient to describe the kinematic motion constraints of interest, whereas the functional description of both the vibratory bowl and APOS feeders required the consideration of kinematic motion constraints over large regions of the CS. On the other hand, the assembly and bowl feeder examples utilized exact integration of motion paths to construct forward projections, whereas the fixture and APOS feeder examples relied on bounded energy models to generate their non-kinematic constraints. This particular set of four example domains was chosen to combine different constraint representations in different ways in an attempt to span the class of problems that might be considered using the motion constraint representations developed.

In each of the four examples, we presented functional metaphors intended to abstract the important relationships between object motions and their constraints without distraction by geometrical or physical details. Of course, these "details" are crucial for ensuring that a particular instance of a system has the desired functional characteristics. In this sense, the configuration space representation is meant to act as a kind of bridge between the abstract function common to all instances or artifacts from a particular domain, and the detailed information that makes each particular instantiation unique. Specifically, the motion constraint representations in configuration space, including the CS, forward projections, and support regions, all possess both the topological properties that map to the abstract functional metaphors, as well as detailed metric information that ensures fidelity with the behavior of the actual example under consideration.

Finally, in each of the examples we attempted to give a sense of how a given system might be modified, or designed, to achieve the desired functional characteristics. Although the motion constraint representations allow us to confirm whether or not a particular system has the desired behavior, and in some cases a sense of how robust that behavior is to potential variations in system parameters, we still do not have an *a priori* means of reliably generating the desired constraints from scratch. This topic will be addressed more fully in the next chapter for the first two examples: peg-in-hole assembly and vibratory bowl feeders. The remaining two examples, although profiting from the developments made for the other examples, await further research into the implementation of conservative and non-conservative energy bounded forward projections.

The purpose of this chapter was to make the representations and visualization techniques introduced in Chapter 2 more concrete by introducing a set of example applications. These examples were chosen to span the available set of constraint representations, as well as to highlight similarities and differences between the various functional constraints. In the next chapter we will consider in detail the *manipulation* of the representations developed so far, and in particular we will apply the resulting tools to the first two example domains introduced in this chapter: peg-in-hole assembly and vibratory bowl feeders.

Design

Chapter 4

You can't always get what you want... But if you try some time, you might find, you get what you need.

– M. Jagger & K. Richards, 1969

In this chapter we will take the representations of function in terms of motion constraint that were developed in the previous chapter and examine how they may be utilized for the purposes of design. We begin by considering a number of potential methods by which objects with the desired constraint characteristics might be generated, and consider a subset of these methods that appear to be both feasible and suitable for design. We then provide an overview of an implemented toolkit for the design of motion constraints in the form of an interactive computer aided design environment. This toolkit is applied to the design of artifacts from two of the example domains in the previous chapter: vibratory bowl feeder tracks and compliant peg-in-hole assemblies. Finally, we discuss some additional characteristics of design using motion constraints and examine the possibility of extending the scope of the toolkit to include fully or partially automated design methodologies.

4.1 The Design of Motion Constraints

The CS, forward projection and support boundary representations in configuration space allow us to perform the analysis necessary to determine if a given system has the desired functional behavior, as was illustrated for the four examples in Section 3. What we lack at this point is the ability to go the other way, that is to create artifacts that exhibit desired functional characteristics. In this section we will consider a number of approaches aimed at inverting motion constraints to produce shapes. We will examine the relationship between design parameters, that describe such things as shape, and the motion constraints that we wish to create. We will extend the scope of functional visualization developed in the previous chapter to capture constraints on the range of admissible variations of design parameters that are consistent with changes made to motion constraints. Finally, we will consider other forms of design operations in addition to the variation of existing design parameters.

4.1.1 Generating Shape from Motion Constraints

We will briefly consider a number of techniques for generating shapes from motion constraints specified in configuration space, and evaluate both their feasibility and suitability for shape design. Considering only kinematics for the moment, one possible formulation of the design problem would be to generate a set of shapes based on a desired set of motions and kinematic motion constraints – i.e. inverting motion constraints to produce shape. As we have seen, the mapping from shape to kinematic motion constraints, as described in Section 2.3, is mathematically well defined and reasonably straightforward to implement. We refer to this as the *forward* mapping from shape to kinematic motion constraints to distinguish it from the *inverse* process: mapping from motion constraints to a pair of shapes. We will briefly consider a few aspects of the inverse problem in order to illustrate that, as might be expected, such a direct inversion is not possible. [1] We will then illustrate an approach, which we refer to as "apparent" inversion, that allows us to perform some limited aspects of such an inversion under a very specific set of constraints.

Direct Constraint Inversion

An intuitively natural approach to consider in generating shape from motion constraints is to construct the *inverse* mapping from a specified set of constraints representing a desired function into shape. Unfortunately, such a direct inversion is not possible for a number of reasons. Consider, for example, constructing an arbitrary CS similar to that in Figure 2.4, but in which some type A facets twist clockwise while others twist in the counter-clockwise direction. Although it would be possible to construct a closed surface consisting of such facets, it would be impossible to generate such a surface with any pair of polygons. It is unlikely that we would construct such an odd, and faulty, CS. Nevertheless, it serves to illustrate the point that there does not necessarily exist a pair of shapes that will generate an arbitrarily constructed surface in configuration space. Put another way, arbitrarily specified constraints may be inconsistent.

[1] In fact, it is not even a good formulation of the design problem, as we shall see shortly.

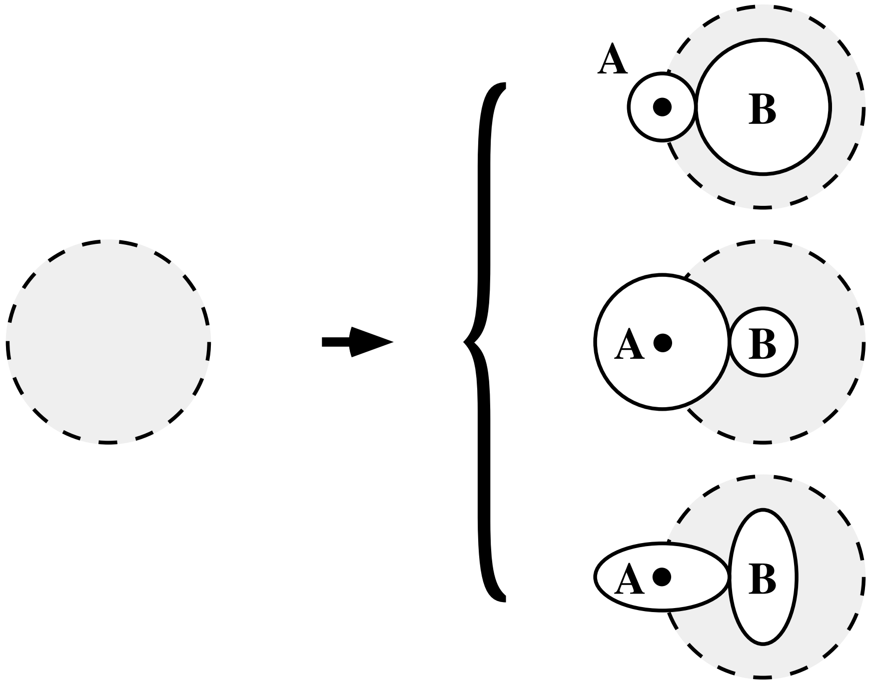


Figure 4.1: A few of the infinite number of shape pairs that produce a circular constraint region in (x, y) configuration space.

Another problem with the direct inversion of constraints has to do with the fact that, even for a *consistent* set of motion constraints in configuration space, there are many (potentially infinite) shape combinations that give rise to identical constraints – the inverse mapping from constraints to shape is underconstrained. Figure 4.1 illustrates this for the simple task of creating a circular constraint region in an (x, y) configuration space. As we can see, there are numerous shape combinations that interact to produce the specified constraint.

Partially Constrained Inversion

Another approach for generating shape from motion constraints that has been suggested by a number of researchers is to constrain the problem by fixing one of the two objects and use it to generate the other object profile so that the desired constraints are achieved. This would be done by taking the fixed shape and sweeping it through space along a predefined motion [51, 38]. The complement of the volume swept out by this motion is a second *maximal* shape that is guaranteed not to interfere with the specified motion – this is a **necessary** condition on the motion constraints. The question that remains is whether or not the resulting shape provides **sufficient** con-

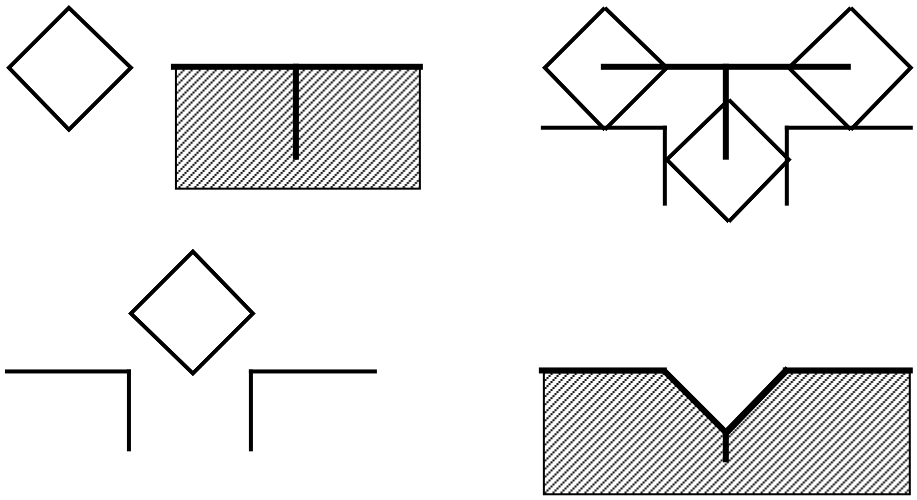


Figure 4.2: An example of the shape generated by sweeping an object along a desired trajectory. The resulting shape interactions do not produce motion constraints consistent with the desired motion.

straints on the motion of the object. Unfortunately, the answer for the general case is no, as can be seen by the example shown in Figure 4.2. In this example the diamond shaped block is guided along a "T"-shaped trajectory with its rotation fixed, sweeping out the volume shown. When the (x, y) motion constraints corresponding to the interaction of the block and the new shape are generated, however, the results are not sufficient to produce only the desired motion. The problem with this approach stems from the fact that, although the shape derived by taking the complement of an object at a given point in configuration space completely constrains that object, sweeping the object along a trajectory may fail because one point along the path can "erase" constraints that were necessary for another point on the path. Specifically, the vertical portion of the imposed "T"-trajectory on the block in Figure 4.2 sweeps away a portion of the shape generated by the horizontal motion. The result is a pair of sharp corners on the new object that, through interaction with the sides of the block, produce the unintentional "chamfers" in the resulting motion constraints.

It is interesting to note that the problem with the approach of generating shape from swept motions is similar to the problem of undercutting common in machining operations. Specifically, if we imagine cutting out a cam profile with, say, a 0.5 inch end mill designed to correspond to the motion of a cam follower of the same diameter, we find that it is impossible to generate a cam profile that will cause the follower to track a curve with a radius of curvature smaller than that of the follower itself. The result of attempting to generate such a profile is the undercut profile shown in

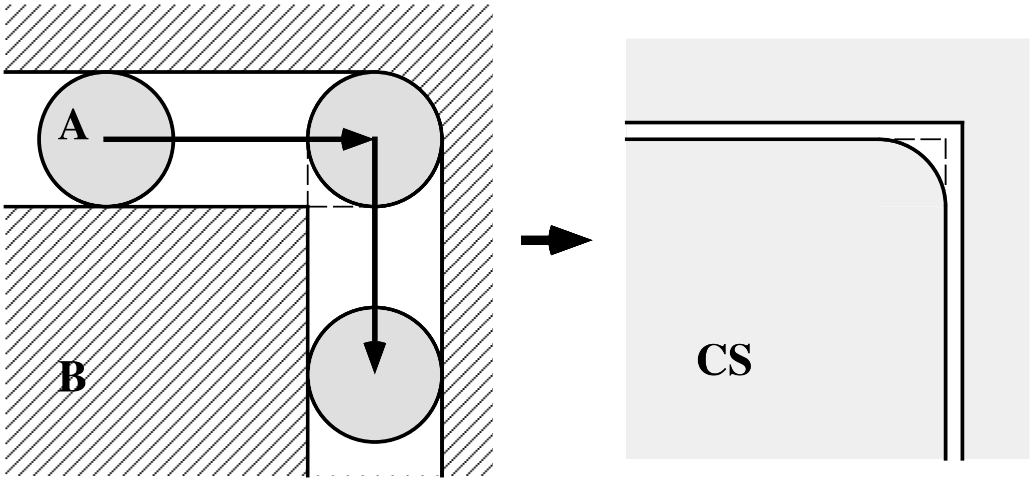


Figure 4.3: An example of undercutting in a cam profile, which is analogous to the inconsistent motion constraints generated in Figure 4.2.

Figure 4.3.

Apparent Constraint Inversion

We have seen that inverting shapes from a specification of desired motion constraints is an ill-posed problem. In the case of direct inversion where both shapes were undefined the inversion from motion constraints to shape was underconstrained. Conversely, for the case of partially constrained inversion where one of the two shapes was specified, the resulting inversion was overconstrained and therefore inconsistent. Beyond these rather significant limitations lies an even more fundamental problem common to both approaches: they assume that we already have a precise specification of the motions and motion constraints that we want to achieve. As we saw in the previous chapter this is often not the case. The motion constraint representations available to us provide a way of recognizing the *class* of motions that are required to produce given functional characteristics, i.e. we often only know what we want when we see it. The question remains, then, as to how we can achieve desired motion constraints as well as the shapes that will produce them. One answer that we propose here is rather simple – we will take advantage of the fact that the forward mapping from shape to motion constraints is well defined by allowing only continuous parametric variations on an existing set of design parameters.

Figure 4.4 illustrates conceptually the process of apparent inversion. Essentially, we start off with a pair of nominal object shapes (planar polygons) from which we generate the CS. We provide a set of a priori mappings between features on the CS

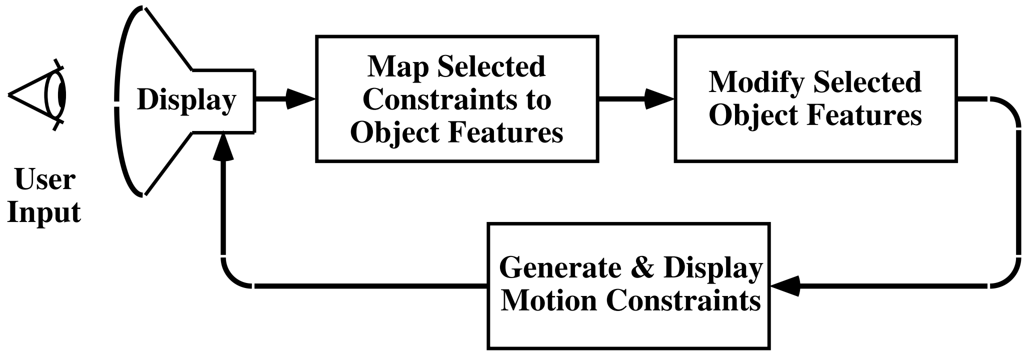


Figure 4.4: A conceptual representation of apparent inversion from motion constraints to shape, where a priori mappings from constraint features to shape features and rapid computation of the motion constraints give the illusion that motion constraints may be inverted to produce shape.

and object features which we then use to **choose shape features in the context of motion constraints**. The parameters describing these object features are then perturbed variationally and the CS for the new objects is computed. If we are able to: *(i)* map parametric perturbations that have intuitive effects on the CS, and *(ii)* perform the computation of the CS rapidly, then as far as the designer is concerned the result is indistinguishable from a true inversion from the constraints to shape. For example, Figure 4.5 illustrates the detailed procedure for apparent inversion of object geometry from the interactive manipulation of a contact constraint facet. Starting in the upper left of the figure where a designer selects a point on the contact constraint surface and displaces it vertically, the apparent inversion algorithm maps the selection to the appropriate shape feature (upper right), modifies the selected shape feature (lower right) and recomputes the new motion constraint surface (lower left). The important point to keep in mind is that the designer sees only the direct manipulation of the motion constraints, as shown in the left half of Figure 4.5. This iterative process is carried out continuously in the background as the designer modifies the constraints.

An important characteristic of the apparent inversion process is that after every iteration the object shapes and the corresponding CS are guaranteed to be consistent because we are actually using the forward mapping from shape to constraints, which is well defined. Of course, nothing is free. What we give up with apparent inversion is the ability to make arbitrary changes to the constraints. Specifically, the arbitrarily imposed a priori mapping from CS features to object features constrains the class of modifications that we may make to the constraints and the shapes. Our task in implementing apparent inversion, such as in Figure 4.5 will be to provide as complete

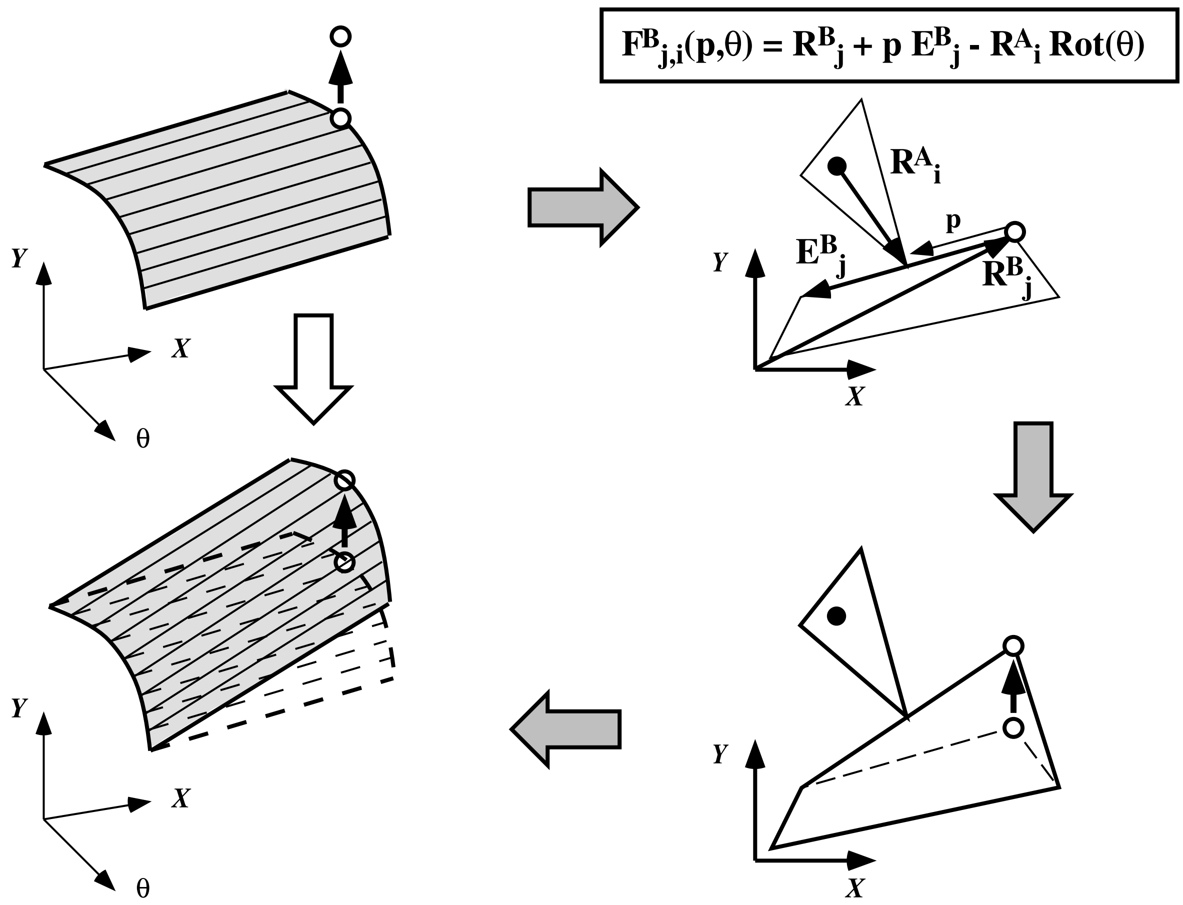


Figure 4.5: Details of apparent inversion for contact constraints → feeder—part shapes.

and flexible set of mappings as possible. The details of this process will be described further in Section 4.2 on design functions.

The motivation for manipulating design parameters by means of apparent inversion from motion constraints is to provide the ability to select and manipulate design parameters directly in the context of function as represented by motion constraints. The basic idea is to be able to grab one or more features on the surface of the CS and simply push or pull them until the constraints have the desired properties, all the while in the background the design parameters are modified so as to be consistent with the new constraints.

4.1.2 The Space of Design Variables

In describing the changes to design parameters and the corresponding changes to motion constraints it is useful to distinguish between the *configuration space* of motions and the *design space* of parameters that define a given system. We distinguish between two classes of design parameters: shape parameters and dynamics parameters. Shape parameters describe the geometry of the objects and may be take the form of a list of polygon vertices represented as (x, y) pairs, as (x, y, z) control points describing cubic polynomial surface patches, etc.. Dynamics parameters include the coefficients of friction and restitution, inertia, gravity and any other non-shape parameters. In the same way that a point in configuration space defines the state of a moving object, a point in design space defines all aspects of a system that may be varied by a designer.[2]

The design space $\mathcal{D}$:

$$\mathcal{D} = P_{shape} \times P_{materials} \times P_{dynamics} \cdots$$

Although an explicit representation of the design space would be prohibitive given the large number of parameters defining a typical system, the concept is useful when considering how changes made to constraint features in configuration space map into changes in the set of design parameters expressed as a point in design space. In particular, we are interested in the behavior of the set of motion constraints in response to a modification of one or more design parameters. A variational modification made to a selected CS feature or set of features may be viewed as an input path specified in configuration space, i.e. a point or set of points selected from the surface of the CS are coerced to follow an imposed input trajectory from their original state to a newly specified state. The changes made to the constraints along this imposed path correspond to changes in one or more design parameters which also may be viewed

[2]For the moment we are neglecting design modifications that would add or remove parameters, such as adding new vertices to the list of vertices defining a polygon. This topic will be addressed in Section 4.1.4.

as a trajectory between start and end states, but in design space. Ideally, there will be a strong correlation between the paths in the two spaces so that, as design parameters are varied by means of apparent inversion, the selected motion constraints will closely follow the desired input trajectory in configuration space.

4.1.3 Dynamic Constraint Visualization

Visualization is important for both analysis and design. In the previous chapter we developed a representation of motion constraints to visualize function from shape interactions. Given a specific design (a single point in the space of design parameters) we were able to determine if that design possessed the desired functional characteristics by visualization of the motion constraints. We refer to this form of visualization as *static* constraint visualization because the parameters and motion constraints remain fixed. In the process of modifying a design, whether by means of apparent inversion from motion constraints or by some other parametric manipulation technique, we are interested in the way these modifications will affect the constraints. We refer to the visualization of the relationship between parametric variations and motion constraint variations as *dynamic* constraint visualization because, unlike static constraint visualization, the motion constraint $\Longleftrightarrow$ parameter relationship is temporal in nature. An important aspect of dynamic constraint visualization is that it takes us beyond simply characterizing the function corresponding to a single point in design space and allows us to explore the *neighborhood* of a design.

The changes imposed on the CS by a designer are local in the sense that they are intended to be made only to the selected CS features. However, local such local changes will typically have very non-local effects as well. For example, each contact facet can be viewed as a dual in the sense that it is determined by a set of parameters describing a pair of interacting features on two different objects, as we saw in Section 2.3.1. The facet equations, derived in Section 5.3.1, may be summarized in the following form:

$$\vec{F}^A_{i,j} = f(\vec{e}^A_i, \vec{v}^B_j) = f(\vec{v}^A_i, \vec{v}^A_{i+1}, \vec{v}^B_j) \tag{4.1}$$

$$\vec{F}^B_{j,i} = f(\vec{e}^B_j, \vec{v}^A_i) = f(\vec{v}^B_j, \vec{v}^B_{j+1}, \vec{v}^A_i) \tag{4.2}$$

where the shape parameters $\vec{v}^B_j$ and $\vec{e}^A_i$ represent the jth vertex and ith edge, respectively, of two interacting polygons. An edge $\vec{e}^A_i$ may be further divided into the pair of vertices $\vec{v}^A_i$ & $\vec{v}^A_{i+1}$ that form its endpoints. As we can see, the characteristics of features from each of the two objects are expressed in every contact facet.

Exactly which CS features will be affected by a local change can be determined from Equations 4.1 and 4.2. For example, a single vertex $\vec{v}^B_j$, of the stationary polygon may interact with all edges $\vec{e}^A_i$, $(i : 0 \rightarrow n)$ of the moving polygon. Any change made to $\vec{v}^B_j$, say during the apparent inversion of a feature on one type A contact facet, will result in simultaneous changes being made to all the other $n-1$ type A facets in which $\vec{v}_j^B$ is a contacting vertex. In addition, from Equations 4.1 and 4.2 we see that $\vec{v}_j^B$ also forms one of the endpoints of both $\vec{e}_{j-1}^B$ and $\vec{e}_j^B$. Therefore, all $2m$ type B contact facets in which $\vec{v}_j^B$ is an endpoint parameter will also change. Therefore, modifying a single vertex on a polygon will change a total $(n+2m)$ contact facets on the CS.[3] As we can see, a considerable degree of coupling exists between design parameters and motion constraint features that are *not* local to only the selected constraint features. Furthermore, this coupling is unavoidable given the nature of the motion constraints forming the CS, and an understanding of the nature of the coupling will be necessary when performing design since a change to a local constraint feature on the CS will bring a number of other, potentially important, CS features "along for the ride".

As an illustration of constraint coupling, Figure 4.6 **(a)** shows a contact facet feature on the CS that has been selected by a designer. Variations imposed on this feature in configuration space are mapped to a single vertex on the stationary polygon in design space by means of apparent inversion. As a point on the selected CS facet is moved along an (x, y, θ) path in configuration space, the corresponding changes to the polygon vertex generated by the apparent inverse operation cause the selected CS feature, as well as other CS features, to change as shown in Figure 4.6 **(b)**.

We will use dynamic visualization as a tool to explore parametric coupling among motion constraints during design. Two properties of apparent inversion that make dynamic constraint visualization possible and useful as a design tool are:

1. **Design Direction:** Parametric $\leftrightarrow$ constraint coupling tells the designer *(i)* what parameters to modify, and *(ii)* which way to go (locally) in design space in order to achieve the desired motion constraints in configuration space.

2. **Consistency Check:** The consistency inherent in the forward mapping constrains the designer to making changes that *are possible* in the context of manipulating motion constraints.

These properties satisfy the necessary and sufficient conditions for getting what you want in a design as well as providing us with the ability to perform "what if" experiments that are a necessary part of the iteration process in interactive design.

A crucial component in dynamic visualization is the ability to modify and reconstruct motion constraints quickly so that the information contained within the relative rates of change among constraints to parametric modifications is accessible to the designer. This will require the ability to compute and display the CS and

[3] Following a similar line of reasoning, changes made to facets also produce changes in the other CS features: edges and vertices. Edges are duals composed of facets, and vertices are triplets composed of edges. As a result, small changes to shape will change facets, edges, and vertices, thus having a potentially profound effect on the overall topology of the CS.

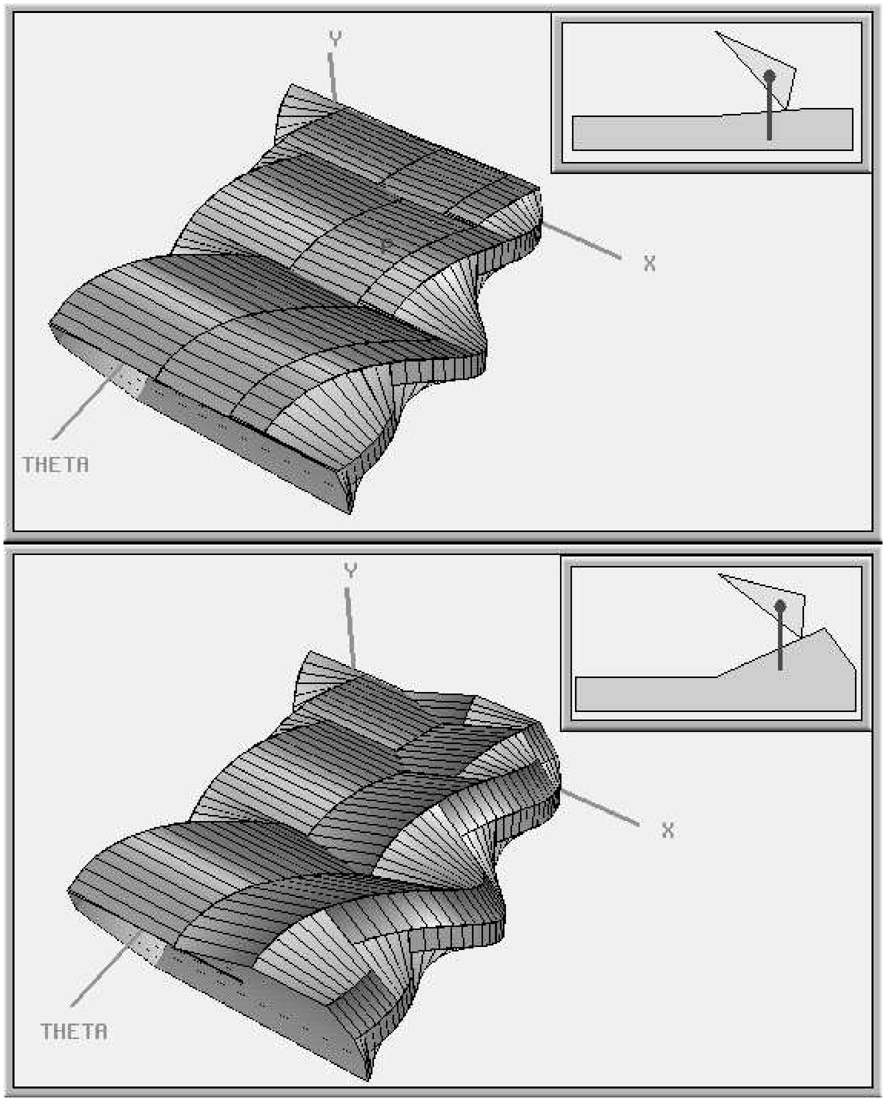


Figure 4.6: An example showing the coupling that exists between features on the CS. A polygon vertex selected and modified by a local point on the CS produces changes that span across a number of other features on the CS.

other motion constraint representations in near real time so that the gap between modification and constraint generation is seamless. For the rest of this chapter we shall assume such computational tools exist. Section 5.8.1 summarizes the kinds of computational structures, assumptions and optimizations necessary to achieve this goal.

4.1.4 Topological vs. Parametric Modifications

In broad terms, the goal of designing motion constraints is to be able to generate motions, or classes of motions, by means of motion constraints. So far we have described a set of ideas, and some tools, that allow us to manipulate the parameters of existing motion constraints. Is this all we can do?

Parametric modifications may be applied only to those parameters that have already been defined, i.e. an existing fixed set of vertices describing a polygon. Five vertices will always describe a five sided polygon, no matter what values their parameters may take. Therefore, an operation such as apparent inversion only allows us to move around within a design space of fixed dimension and topology. Topological modifications, by which we mean changes to the size and topology of the design space, introduce new design parameters (or eliminate old ones) and fundamentally change the class of artifacts that may be generated. For example, generating new shapes by sweeping an object along a fixed path in configuration space, discussed in Section 4.1.1, is a topological modification. As we noted in Section 4.1.1, we cannot in general guarantee that the constraints produced by such a topological modification will be consistent with the intended motion. There are, however, some swept motion operations where this may be done consistently, one of which will be discussed in Section 4.2.2. There are still other topological design operations that may be performed consistently outside the context of constraint inversion. For example, the initial specification of nominal designs upon which parametric modifications will be used to iteratively arrive at a suitable design. Such nominal designs may be generated or selected from a taxonomy of possibilities, as discussed in Section 4.4.4.

Such topological modifications are attractive because they allow us the luxury of "jumping around" within and between multiple design spaces, as opposed to moving variationally within a fixed design space that may, or may not, contain a valid solution point. At the same time, we have already seen how local changes to a few existing parameters in a fixed design space may have significant non-local effects on the motion constraints in configuration space. We can only imagine what effect adding and deleting whole sets of parameters will have.

Comparing the relative strengths and weaknesses of parametric and topological modifications of motion constraints we have:

Parametric Modifications:

(+) The ability to generate shapes from constraints consistently.

(+) The ability to explore the neighborhood of a point in design space and to observe coupling between parameters and motion constraints.

(-) Limited to varying existing parameters that correspond to only one class of designs.

(-) Constrained to making incremental motions in a fixed design space. Suitable designs may be "distant" from a given nominal design, and a solution may not exist within the given set of design parameters.

(-) Reliance on the expected *smoothness* of design space during variational modifications. it is possible that for some cases a suitable design may exist only within a very small region that could be bypassed while varying parameters.

Topological Modifications:

(+) Potentially a great deal of flexibility in exploring different design possibilities encompassing many classes of designs (i.e. many design spaces).

(+) The ability to "jump" between designs that are distant in terms of design parameters, or are not contained within the same design space.

(-) Difficult (often impossible) to perform consistently in the context of inverting motion constraints.

(-) A very limited set of viable design techniques exists for a few application domains.

(-) Adding and removing design variables, or otherwise changing the topology of the design space, can make it difficult to close in on a viable design. Jumping around within a design space may cause us to miss potentially valid designs, and generally makes a methodical search of possible designs difficult to perform. Adding design variables invariably increases the dimension of the design space, also adding to the overall complexity of the design task.

As we can see, the strengths and weaknesses of parametric and topological modifications are more or less complementary in nature. Basically, we would like as much flexibility as possible while at the same time control the complexity that we must deal with. This suggests that a good design strategy would be to combine the best of both approaches wherever possible. Specifically, we will want to take advantage of the flexibility of topological modifications early on in a design to "jump" among possible nominal design topologies until we find what appears to be a promising class of designs. Then, when we are near what we hope is feasible design, we may more methodically explore the local parameter space for possible solutions. Section 4.4.5 describes one methodology appropriate for designing vibratory bowl feeders.

4.1.5 Interactive vs. Automated Design

In our discussion so far we have assumed that the modifications necessary to generate a design are to be carried out by a human designer working interactively with a representation for visualizing, and set of tools for manipulating, the motion constraints that describe function. What about automated design? We are able to automatically generate representations of function in terms of motion constraints given geometry and other parameters as described in Chapter 2 and detailed in Chapter 5. These constraints are mathematically precise, and because they are computer generated they are also computationally accessible. Therefore, it would seem that we should be able to manipulate these representations automatically using such tools as apparent inversion in order to perform design.

The kinematic and dynamic constraints that we consider explicitly in the motion constraint representation are only part of a much larger set of potential design constraints, i.e. cost, machinability, maintainability, etc.. Automating these portions of the design task will almost certainly result in the generation of many useless designs.[4] The human designer, on the other hand, can keep more of constraints in mind while using this tool. In this research we focus on the interactive/iterative design paradigm because it provides us with more flexibility and is generally a more tractable approach to design. We will therefore focus on the role of the computer as a tool for automating the task of generating and displaying the explicit representations of motion constraints, and allowing us to interactively manipulate both the constraints and the parameters defining them using such tools as apparent inversion. This emphasis on interactive design will also require us to produce an implementation that is fast and efficient in computing the necessary representations. Later, as we gain insight into the relationships between parameters and constraints, and the more general relationship between function and motion constraints, these tools will also be useful in developing the additional representations and algorithms necessary to support automated and semi-automated design, which we will briefly discuss in Section 6.2.3.

4.2 Design Functions

Our goal is to have consistent modifications of motion constraints mapped into the appropriate parametric modifications. Recalling the space of design parameters discussed in Section 4.1.2, variational modifications to design parameters may be viewed as a path between states in design space. In this context, we view design as one or more functions mapping desired changes to motion constraints, expressed as paths

[4] By the same token, a system that automatically generates large sets of design alternatives, even though many are infeasible for one reason or another, may have value as an aid to human designers. See Ulrich [75] for examples of this approach.

imposed by a designer on a representation of motion constraint features, into parametric changes represented as a path in design space. More specifically, *design functions* are mappings from user inputs that *(i)* select, and *(ii)* modify the appropriate design parameters in the context of motion constraint modifications.

Although the above description implies that design functions are intended purely for modification of existing parameters, we may expand the notion of design functions to include those topological modifications to motion constraints that may be performed consistently. In this section we will examine examples of both parametric and topological design functions. We will first consider parametric functions to implement apparent inversion for two forms of motion constraint representations: contact facets and planar support constraints. We will then describe a topological design operator for generating classes of support polygon geometries and discuss why it may be implemented consistently.

4.2.1 Apparent Inversion of Motion Constraints

There are three distinct functional components of apparent inversion from motion constraints:

1. **Parameter selection:** Select parameter(s) that are to be varied from the constraint representations.

2. **Input mapping:** Map the designer's intended modification to the selected parameters into a path in (x, y, θ) configuration space.[5]

3. **Parametric mapping:** Map the (x, y, θ) path to a path in design space, specifically to the selected parameter(s), by inverting the corresponding constraint expressions.

More precisely, these operations may be expressed in terms of the following functions:

$$\mathcal{F}^{select}\left(P_{CS}^{(x,y,\theta)}, \vec{F}^{A/B}\right) \rightarrow s \tag{4.3}$$

where $P_{CS}^{(x,y,\theta)}$ is a point selected from the surface of the facet $\vec{F}^{A/B}$, and s is a parameter or set of parameters from the facet's describing equation. For the input mapping, we have:

$$\mathcal{F}^{input}\left(\mathcal{P}_{input}\right) \rightarrow \mathcal{P}_{CS}^{(x,y,\theta)} \tag{4.4}$$

[5]Input mapping is an artifact of the type of input device used. Typically the input will in the form of an offset in the screen coordinates of a cursor via a mouse. It is necessary to map such a two variable input into a motion in the three-dimensional configuration space. If a three d.o.f. input device is used, then the input mapping function may be unnecessary.

where $\mathcal{P}_{input}$ is an input path in whatever space the designer interacts with the constraint representations, and $\mathcal{P}_{CS}^{(x,y,\theta)}$ is the corresponding path in configuration space. And finally we have:

$$\mathcal{F}^{modify}\left(s, \mathcal{P}_{CS}^{(x,y,\theta)}\right) \rightarrow \left(\forall \vec{v}_i^{polygon} \in s, f^{-1}(\vec{v}_i^{polygon})\right) \tag{4.5}$$

which applies the input path in configuration space to the selected parameters, which are assumed here to consist of polygon vertices.

The above functions are designed to operate on parameters describing polygon geometry in terms of (x, y) vertices, and motion constraints represented in (x, y, θ) configuration space. In this research we have developed and implemented specific apparent inversion functions for two constraint representations: the contact facets of the CS and the support transition boundaries on the surface of the CS. These parameters and representations were chosen because of their prominence in determining the function of vibratory bowl feeders and compliant assemblies described in the previous chapter. Similar functions could also be generated to operate on other design parameters, such as dynamics and material properties, as well as manipulating parameters within the context of other constraint representations, such as the forward projections including discrete paths and bounded energy regions. Such additional design functions have not been detailed or implemented in this report.

Inversion of Contact Facet Constraints

The kinematic constraint representations to be manipulated from the CS are the individual contact facet equations 4.1 and 4.2 discussed in Section 4.1.3, and derived in detail in Section 5.3.1. To modify a contact facet, the designer selects a point on the surface of a facet. The selected point is mapped to a polygon vertex by means of the selection function given by:

$$\mathcal{F}_{CS}^{select}\left(P_{CS}^{(x,y,\theta)}, \vec{F}^{A/B}\right) = \vec{v}_i^{A/B} \in \vec{F}^{A/B}$$

where $P_{CS}^{(x,y,\theta)}$ and $\vec{F}^{A/B}$ are the selected point and facet as described in Equation 4.3, and the selected parameter $s = \vec{v}_i^{A/B} \in \vec{F}^{A/B}$ is a vertex from one of the two polygons forming the facet. By default, the edge vertex closest to the (x, y) component of the selected $P_{CS}^{(x,y,\theta)}$ point is chosen. Therefore, for a type A facet, the vertex will be from polygon A, and vise versa. This a priori selection is, of course, totally arbitrary and could be user selected.

The input mapping function is given by:

$$\mathcal{F}_{CS}^{input}\left(\mathcal{P}_{input}\right) = \delta(x, y)$$

where $\mathcal{P}_{input}$ is a path in the form of an offset in the screen coordinates of a cursor moved by a mouse. The θ component of $P_{CS}^{(x,y,\theta)}$ computed from $\mathcal{F}_{CS}^{select}()$ is used to define an (x, y) plane through that point. As the mouse is moved, the intersection of a line projected into configuration space from the mouse screen coordinates determines the corresponding offset $\delta(x, y)$ representing the path $\mathcal{P}_{CS}^{(x,y,\theta)}$ in configuration space.

Finally, the inverse mapping function is:

$$\mathcal{F}_{CS}^{modify}\left(\vec{v}_i^{A/B}, \delta(x, y)\right) = \vec{v}_i^{A/B} + \begin{cases} \delta(x, y) & \text{if } \vec{F}^B \\ -Rot(\theta)^{-1}\delta(x, y) & \text{if } \vec{F}^A \end{cases}$$

where, depending on the type of polygon to which each vertex corresponds, the offset $\delta(x, y)$ is added to the selected vertices $\vec{v}_i^{A/B}$ with or without being mapped to the appropriate θ orientation by $Rot(\theta)^{-1}$. The purpose of the $-Rot(\theta)^{-1}$ mapping is to ensure that variations to polygon A vertices cause any selected points on the type A contact facets to follow the input path $\delta(x, y)$. Multiple vertices may be selected with $\mathcal{F}_{CS}^{select}()$ before invoking the function $\mathcal{F}_{CS}^{modify}()$.

In Section 4.1.3 we discussed the inherent coupling between modifications made to a single polygon vertex and the resulting changes to a number of contact facets in the CS. Given the specification of the above design functions, what more can we say about the specific nature of this coupling during apparent inversion? As we noted earlier, contact facets are formed by the interaction of an edge feature of one polygon and a vertex of the other. From the inverse mapping function $\mathcal{F}_{CS}^{modify}()$ we can see that a variation of $\delta(x, y)$ applied to a vertex of the stationary polygon B will *shift* the portion of each facet containing that vertex, either as a contacting vertex or as the endpoint of a contacting edge, by a constant amount. More importantly, this shift will be invariant in (x, y) across the range $\theta = 0 \rightarrow 2\pi$. A variation of $\delta(x, y)$ applied to a vertex of the moving polygon A, however, will produce an offset in configuration space whose (x, y) orientation varies as a function of θ, specifically $-Rot(\theta)^{-1}\delta(x, y)$. The resulting change will be in the form of an expanded (or contracted) helix. Figure 4.7 illustrates these effects on a space curve describing a vertex-vertex contact between polygons in (x, y, θ) configuration space to an (x, y) offset in the moving polygon A's vertex and to the stationary polygon B's vertex.

The space curve formed by the vertex-vertex contact illustrated in Figure 4.7 is equivalent to the adjacency boundary between two facets. The effect that the above transformations will have on an entire contact facet formed by a vertex-edge contact will depend on whether the vertex being modified is the contact vertex or an endpoint of the contact edge. Figure 4.8 illustrates the effects for the modification to a moving polygon A vertex that is **(a)** the contact vertex, **(b)** an edge endpoint vertex, and for the modification to a stationary polygon B vertex that is **(c)** the contact vertex, and **(d)** an edge endpoint vertex. We note that by a priori mapping a selected point $P_{CS}^{(x,y,\theta)}$ from a facet $\vec{F}^{A/B}$ to an endpoint vertex $\vec{v}_i^{A/B}$ on a polygon edge, we are constraining the designer to making only the modifications shown in Figure 4.8 **(b)** and **(d)**.

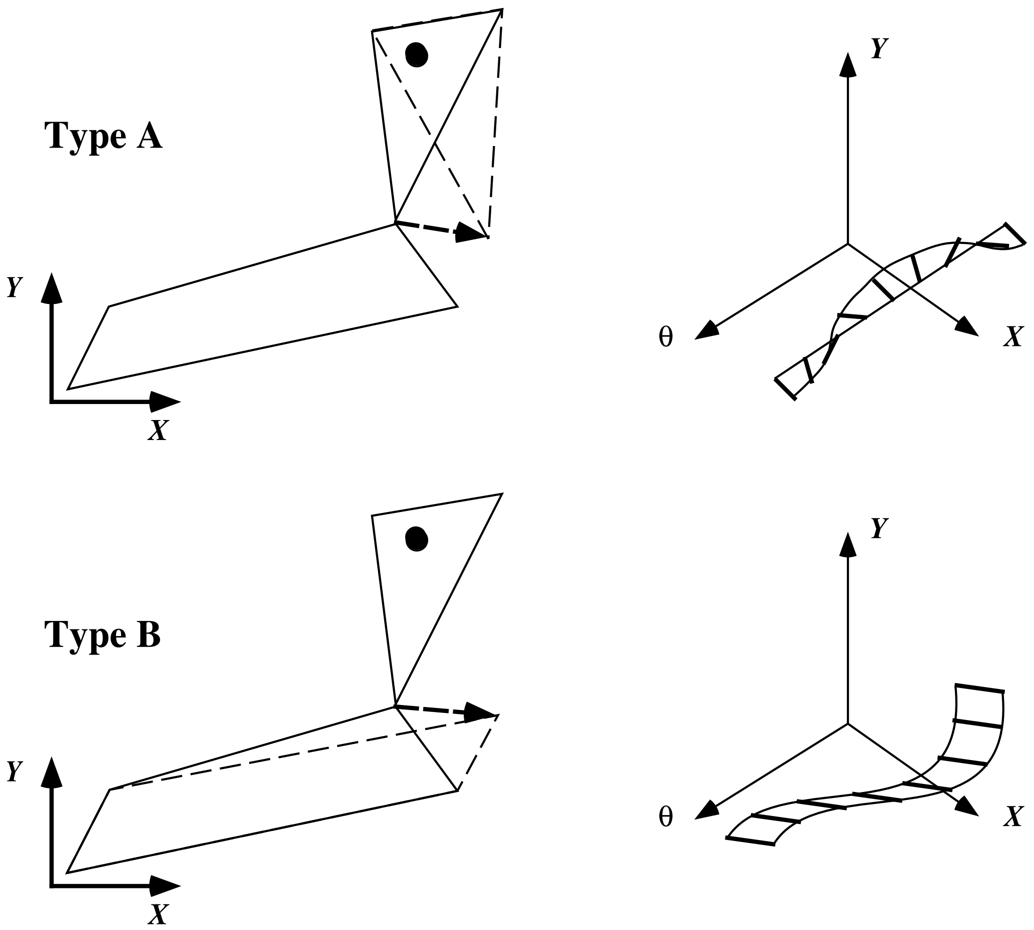


Figure 4.7: The effect of changing a moving polygon A vertex and a stationary polygon B vertex to the vertex-vertex contact constraint in (x, y, θ) configuration space.

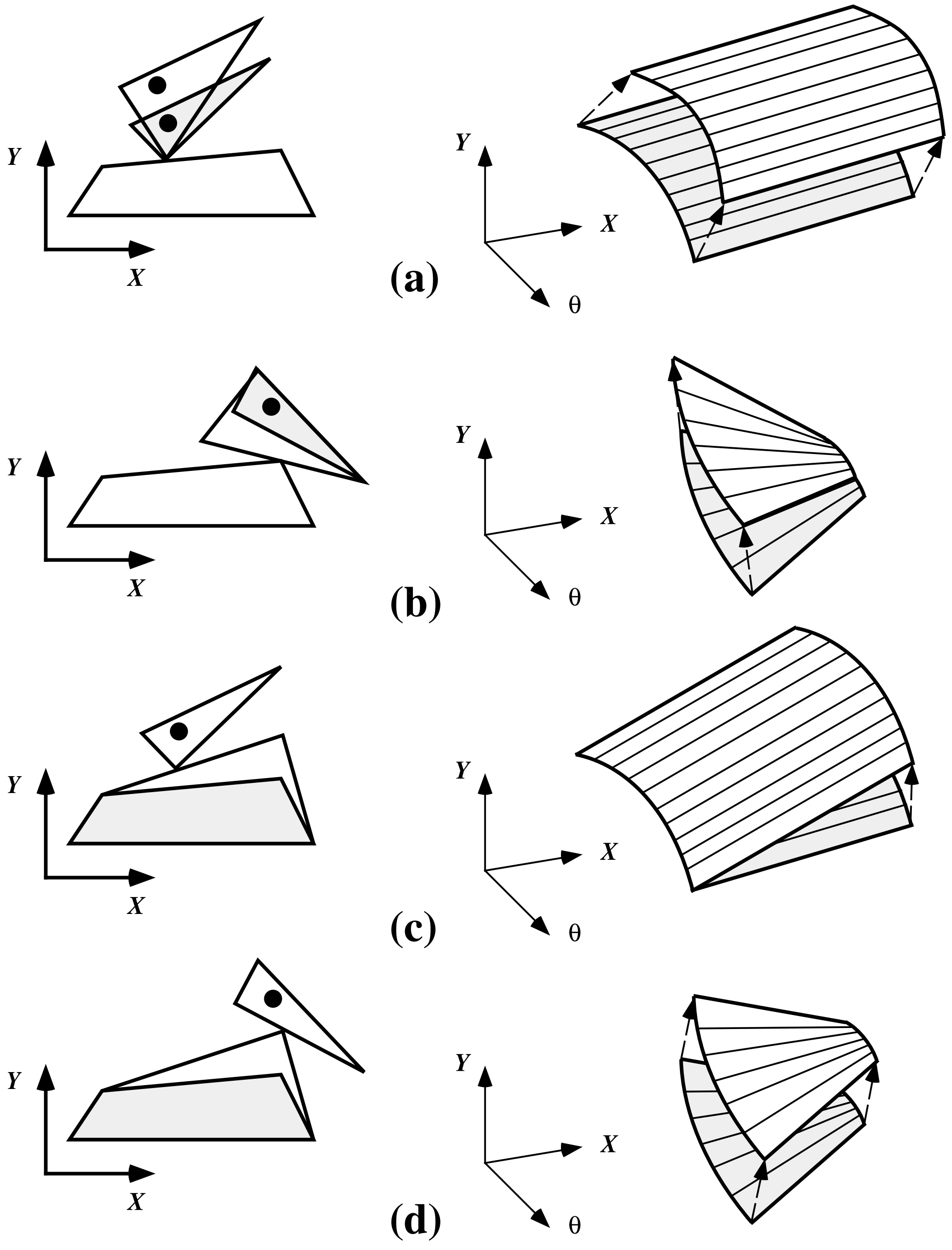


Figure 4.8: The effect of changing a moving polygon A vertex that is **(a)** the contact vertex, **(b)** an edge endpoint vertex, and a stationary polygon B vertex that is **(c)** the contact vertex, and **(d)** an edge endpoint vertex.

Inversion of Support Transition Boundaries

The constraint representations to be manipulated from the CS are the support constraint boundaries that intersect the CS surface. As described in Section 2.5, these boundaries partition the surface of the CS into supported and unsupported contact configurations. To modify a region, the designer selects a point on one of the boundaries, which represents an (x, y, θ) position where the moving object is marginally supported by the supporting polygon. The selected point maps to a series of support polygon vertices as described by:

$$\mathcal{F}_{support}^{select}\left(P_{CS}^{(x,y,\theta)}, \vec{F}^{A/B}\right) \rightarrow \left\{s \subset \vec{v}_i^{track},\ i = i, n\right\}$$

where $P_{CS}^{(x,y,\theta)}$ and $\vec{F}^{A/B}$ are the selected point and the selected contact facet, respectively, and s is a set of vertices from the supporting polygon. The vertices contained in s are determined by examining those vertices and edge segments of both the moving and supporting polygons that act to support the moving object at that (x, y, θ) position in order to determine the subset that are critical to the support. Specifically, the critical support points will all lie along a line, passing through the **cg** of the object, on one side of which lie all the remaining support points. This line forms the axis about which the moving object will rotate out of the (x, y) plane as it falls off the supporting polygon. Section 5.5.2 provides a more detailed treatment of the planar support computation.

The input mapping function is:

$$\mathcal{F}_{support}^{input}\left(\mathcal{P}_{input}, \vec{F}^{A/B}\right) = \left\{\delta(x, y) : \delta(x, y) \parallel \vec{F}^{A/B}, \theta_{fixed}\right\}$$

where, as before, $\mathcal{P}_{input}$ is a path in the form of an offset in the screen coordinates of a cursor positioned by means of a mouse. Unlike the input mapping function for CS inversion, the offset corresponding to a motion of the cursor in screen coordinates is mapped to a *line* parallel to the surface of the contact facet $\vec{F}^{A/B}$ at the selected point $P_{CS}^{(x,y,\theta)}$ and perpendicular to the θ axis of the configuration space. The corresponding offset $\delta(x, y)$ along this line forms the input path $\mathcal{P}_{CS}^{(x,y,\theta)}$.

The inverse mapping function is given by:

$$\mathcal{F}_{support}^{modify}\left(s, \delta(x, y)\right) \rightarrow \vec{v}_i^{track} += \left\{\delta(x, y) : \forall \vec{v}_i^{track} \in s\right\}$$

where the offset $\delta(x, y)$ is mapped to the critical support polygon vertices $\vec{v}_i^{track}$ contained in s. The result of this mapping is to move the critical support vertices $\vec{v}_i^{track}$ in such a way that the support transition boundary local to the selected point $P_{CS}^{(x,y,\theta)}$ moves along the surface of the contact facet $\vec{F}^{A/B}$ in unison with the specified input path $\mathcal{P}_{CS}^{(x,y,\theta)}$.

Figure 4.9 shows the inverse mapping function applied to a point on the boundary of an unsupported region on the CS, where the support polygon vertices determined to be critical by $\mathcal{F}^{select}_{support}()$ are highlighted. We notice that as the selected boundary point on the contact facet is moved between frames **(a)** and **(b)** that the selected support polygon vertices are shifted, changing the entire boundary's position and shape. This is yet another illustration of the coupling between constraints and design parameters, in this case between the support transition boundaries and the support polygon vertices. As we will see in Section 5.5, we do not have available to us analytic functions describing these boundaries on the CS, and hence have no direct means of determining their behavior in response to design modifications. By means of approximate numerical computations built into the apparent inversion operation we are able to empirically observe the highly nonlinear coupling and "experiment" with different modifications, one example of which is shown in Figure 4.9. As with the apparent inversion of the contact facets on the CS, the speed of computing the support transition boundaries is critical in making the inverse mapping function useful for design.

The above design functions for the apparent inversion of contact facets and support transition boundaries represent approximations to the more general design functions outlined in Equations 4.3, 4.4 and 4.5. Although the approximations in some cases restrict the class of modifications possible for a single design manipulation step, they are intended to at least span the set of design variables so that a suitable parametric design, if one exists, may eventually be found. In general, this will be accomplished by the iterative use of a combination of the above design functions that allow the us to navigate our way through design space using the motion constraint representations as a guide. The actual implementation of these functions is described in more detail in Chapter 5.

4.2.2 Out-of-Plane Swept Geometries

In Section 4.1.4 we discussed some of the relative advantages and disadvantages of parametric vs. topological constraint modification operations. Where applicable, topological operations give us the ability to introduce entirely new classes of design solutions, along with new design parameters. Consider the vibratory bowl feeder example in Section 3.2, where one of the parameter sets used to determine the feeder-filter function was the shape of the supporting track. As we saw above, the coupling between the support transition boundaries and the supporting track polygon vertices is very pronounced and non-linear. In trying to terminate a given part motion path on the CS by having it encounter an unsupported region, it would be tempting to simply place such a region in front of the path by generating the appropriate track geometry instead of wrestling with the constraints imposed by the existing track geometry. In fact, we can consistently define such a function, which we refer to as the *cutout function*.

Figure 4.10 **(a)** illustrates a polygonal part in the plane. We begin by selecting

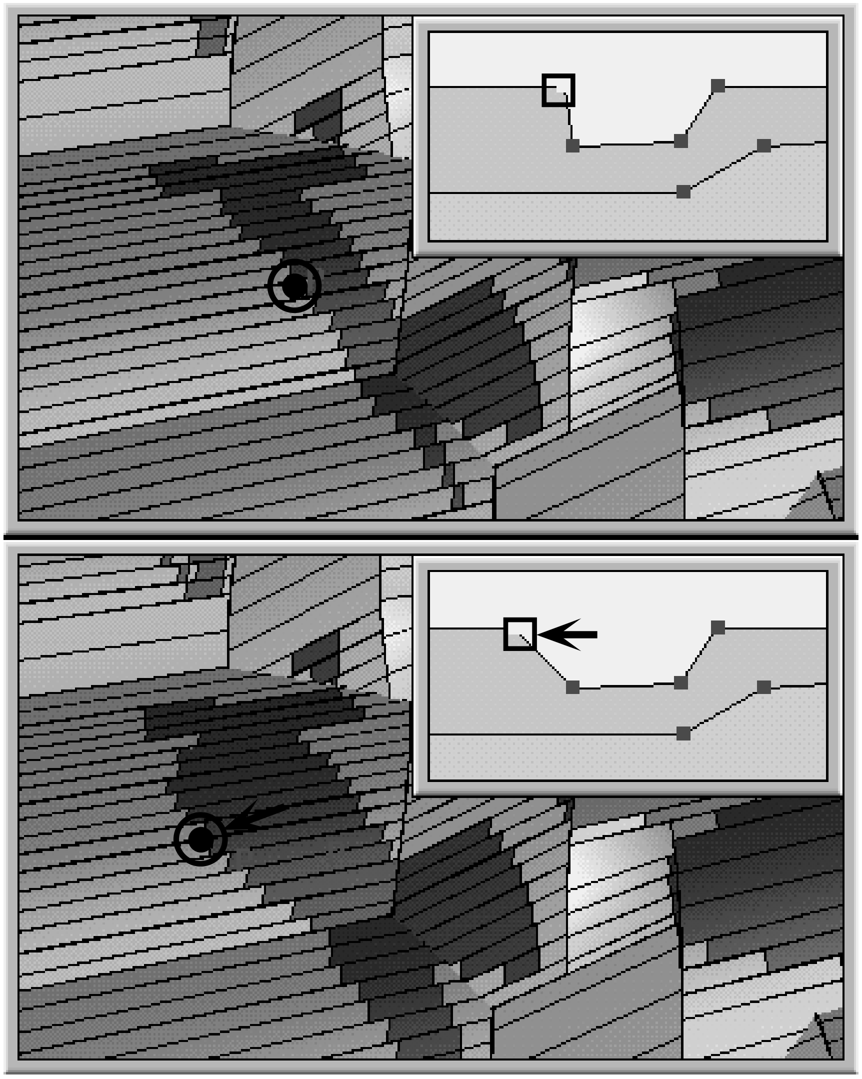


Figure 4.9: Apparent inversion of a support transition boundary on the surface of the CS.

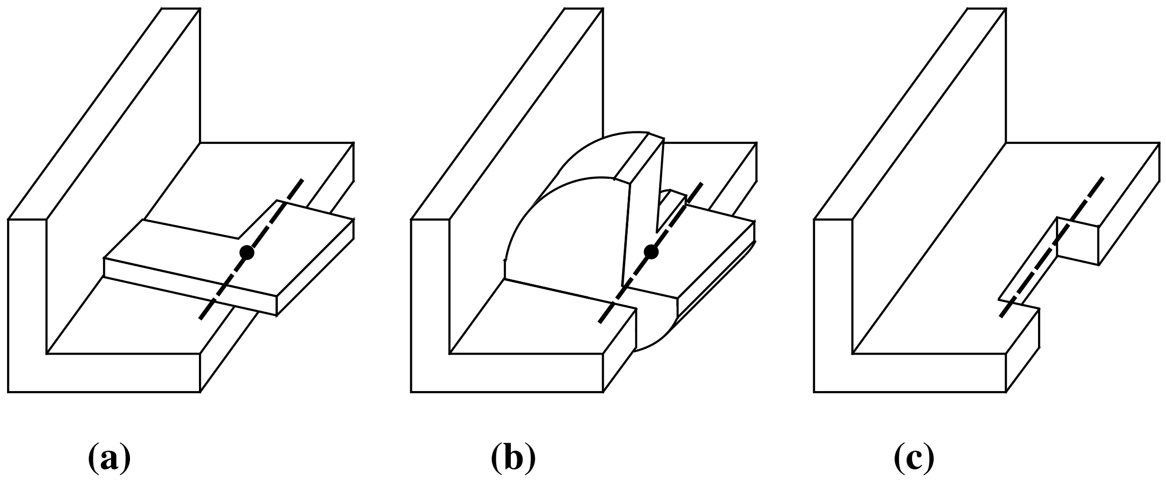


Figure 4.10: Generating a support track cutout.

an (x, y, θ) configuration of the part where we wish it to transition from a supported to an unsupported state. Next, we specify the direction in which we want the part to fall by defining an oriented line through the part **cg**. This line represents the axis about which the part will rotate out of the (x, y) plane at the given position. To generate the track contour that will allow this out-of-plane motion of the part we generate a shaped cutout from the track corresponding to the part contour on the *unsupported* side of the fall axis, as shown in Figure 4.10 **(c)**. We can think of the cutout function as using the part as a sort of "can opener" by rotating the part about the fall axis to sweep out a portion of the track **(b)**.[6]

The above description of the cutout function deals exclusively with the part and track geometries, whereas we have been focusing our attention on the motion constraint representations to describe function. What does the cutout function look like in terms of the support transition boundaries on the CS in configuration space? Figure 4.11 shows the result of adding a cutout at an (x, y, θ) point on a CS facet. In the ideal case, the unsupported region generated by the cutout would appear simply as a single point or small region on the contact facet surface at the desired (x, y, θ) configuration. In reality, however, a number of additional non-local unsupported regions are also introduced on the CS surface. These additional unsupported regions arise from the simple fact that a hole generated for a part to drop through in one orientation does not, in general, prevent parts in other orientations from falling

[6]As described in Section 5.7, the implemented cutout function generates a convex approximation to the cutout contour described here. In addition, to preserve the genus-zero topology of the supporting track polygon, the cutout is connected to the outer track contour where necessary.

through as well. The location, number, size, and shape of these additional regions are determined by the location and shape of the track cutout contour, which in turn is determined by the chosen (x, y, θ) configuration and fall axis orientation for the cutout. As expected, the topological modifications created by the cutout function add complexity as well as flexibility to the design task by introducing a number of new polygon vertices. Controlling this additional complexity while at the same time taking advantage of the ability to introduce an unsupported region at any arbitrary location on the surface of the CS is the major challenge of utilizing the cutout function as a design tool.

We noted in Section 4.1.1 that the shape created by sweeping out a part along a desired path in configuration space satisfied the necessary conditions, but not the sufficient conditions for the desired motion constraints along that path. Does the cutout function above satisfy both the necessary and sufficient conditions for producing the desired motion constraints, and if so, why? We recall from Section 4.1.1 that the problem with the swept shape generation approach was the fact that the volume swept out by the fixed object on one portion of the specified path could eliminate shape features that were necessary to constrain the object along another portion of the path, as illustrated in Figures 4.2 and 4.3. If we imagine the rotation of the part out of the (x, y) plane as a path in a higher dimensional configuration space, then only the point corresponding to the *beginning* of the path is contained within the lower dimensional (x, y, θ) slice of that space. As a result, the support constraints provided by the track at the (x, y, θ) position where the out-of-plane path begins are not affected by any other point along the remainder of that path. By this argument, the shape generated by the swept out-of-plane motion provides the intended motion constraints, i.e. support constraints, at the selected (x, y, θ) point. We are, however, ignoring an important additional factor – we neglect the remainder of the part motion outside the (x, y, θ) plane. In particular, we could imagine a case where the half of the part on the supported side of the fall axis is *wider* than the unsupported half used to generate the cutout contour. In this case, a part may rotate out of the plane as desired, but become caught in the cutout rather than falling off the track. In this research we are assuming that the parts are thin enough compared to their (x, y) dimension that we could, if necessary, cut an additional narrow *slot* along the fall axis whose length exceeded the widest cross-section of the part. This slot would be swept out by the part as it reached a vertical orientation and then slid downward in the $-z$ direction. Although an inelegant solution, such a slot would allow the part to slide off the track while at the same time yet be narrow enough so as not to compromise the support characteristics of parts in other orientations in the plane.

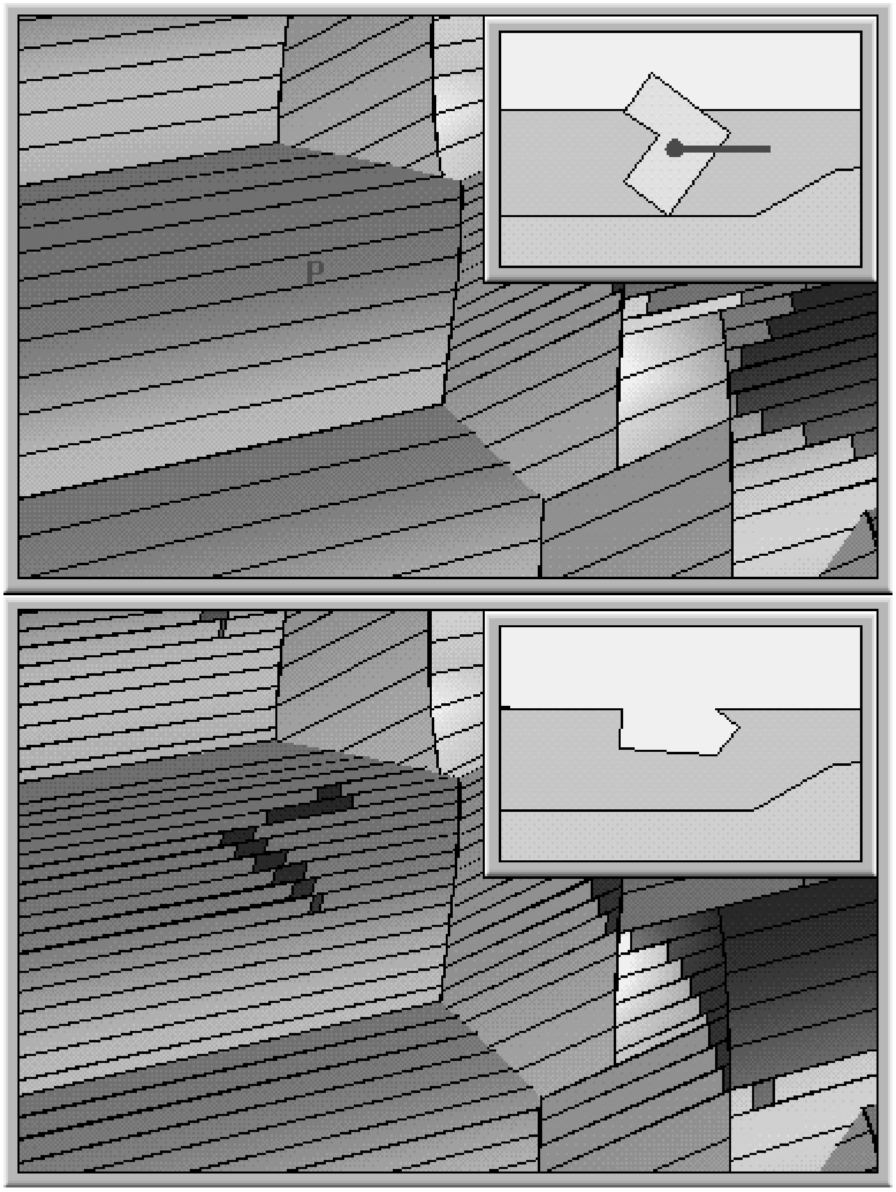


Figure 4.11: Multiple unsupported regions generated on the CS by the cutout function applied to a single (x, y, θ) configuration (indicated by the "P" in the top figure).

4.3 A Toolkit for Visualization and Design

In this section we give a brief overview of an implemented set of tools that form a computational environment for visualization and design called `cspace-shell`. Specifically, we've selected the representations for the CS, numerically integrated forward projections, and unsupported regions on the CS surface, which are a subset of the motion constraint representations discussed in Chapter 2. This subset was chosen because it will be sufficient to represent both the compliant peg-in-hole assembly and vibratory bowl feeder examples described in Section 3.

4.3.1 Assumptions

Most of the following the assumptions made in computing motion constraints have been discussed in earlier chapters but are repeated here for completeness. Where appropriate, we note assumptions that are unique to specific application domains.

- Objects are modeled as infinitely rigid planar polygons that do not change shape under applied loads.
- The dynamics governing object motions are assumed to be quasi-static: body forces due to velocities and accelerations of the objects are considered negligible in comparison to static forces.
- Externally applied forces, such as gravity, are assumed to act through the reference point of the moving object. For the vibratory bowl feeder example, we assume this point to be the object's center of gravity **cg**, whereas for the compliant assembly example the reference point is the remote center of compliance (RCC).
- The contact between the moving object and plane of motion is assumed to be frictionless: reaction forces occur solely between polygon *boundaries* in the plane of motion.
- In the vibratory feeder examples, we assume that parts move along the track individually, with contacts occurring only between the part and feeder wall – no stacking or bunching of parts is considered.
- Out of plane motions, such as hopping of parts on a vibratory feeder track, are assumed to be negligible in comparison to in-plane motions.
- Parameters are known exactly, and part motions are modeled as being completely deterministic.

4.3.2 Overview and Layout

The main functions contained within `cspace_shell` are:

- **Interactive GUI** – mouse based input via buttons, sliders, and surface point selection.
- **Geometric modeler** – representation and manipulation of planar polygons.
- **Motion constraint visualizer** – engine for rapid computation, display, and interrogation of motion constraint representations.
- **Apparent inversion functions** – parametric selection and manipulation via constraint representations.
- **Parametric coupling visualizer** – highlighting of motion constraint features coupled to selected polygon vertices.
- **Functional verification** – numerical simulation and animation of part motions.
- **UNDO!** – can play with shapes and constraints without destroying existing designs.
- **Feeder path optimizer** – rendering of motion path thickness is proportional to the probability of part entering a feeder in a given orientation. Feeders passing thicker paths have a higher average throughput.

Figure 4.12 shows the layout for the main interface with `cspace-shell`. The large window on the left displays the various motion constraints in (x, y, θ) configuration space. The viewing angle and zoom for this window are controlled by the view buttons in the lower left. The three smaller windows on the right are from top to bottom: the display for the moving polygon, the display for the stationary polygon (and track, if included), and the object position and animation display window. The sliders in the bottom center control the resolution to which the approximate unsupported regions are computed on the CS, the direction of the applied force to the moving polygon, the radius of gyration of the moving polygon, and the coefficient of friction between the moving and stationary polygons. The two small radio buttons above the top right of the large motion constraint viewing window allow the user to select between apparent inversion of *(i)* the contact facets forming the CS, and *(ii)* the support transition boundaries. The buttons across the top of the figure control various other functions including: file I/O, mode selection, undo of last modification, cutout function selection, motion path computation, and path animation. There are two other windows (not shown) that may be selected via the mode selection menu. The first is a rendering control window containing a palette of 32 colors that are

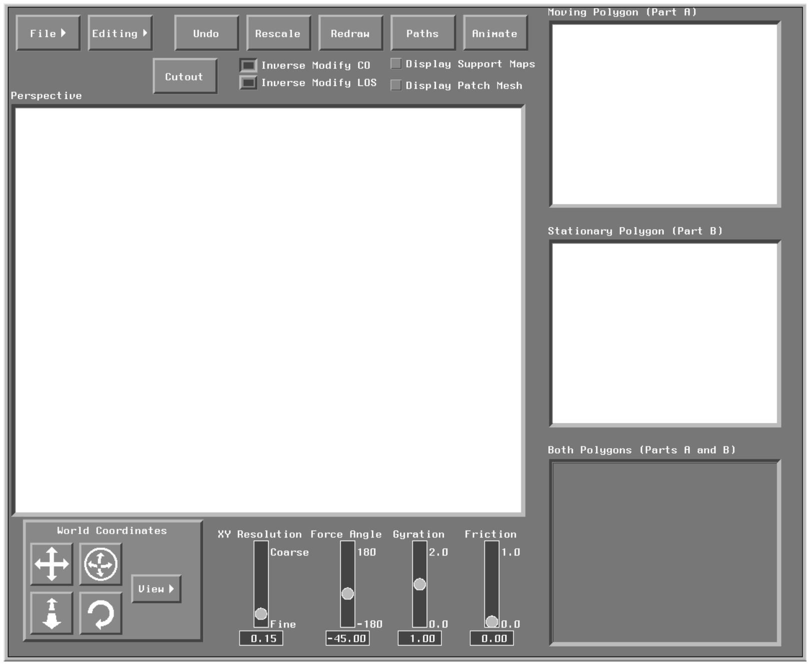


Figure 4.12: Main graphical interface for `cspace-shell`.

varied by means of an HSV color wheel, and surface properties for the lighting model (i.e. shininess, specularity, etc.) that are varied by sliders. These colors and properties are used to display the objects and constraint surfaces. The second window contains sliders and buttons for modifying the settings for the apparent inversion design functions, the cutout function, and for setting the initial positions for motion paths in compliant assembly.

4.4 The Design of Vibratory Bowl Feeders

To demonstrate the use of the representations and manipulation tools described above, we will now return to the vibratory bowl feeder example introduced in Section 3.2.

4.4.1 Design Parameters and Constraint Representations

Figure 4.13 shows the 3D polyhedral and corresponding planar models introduced in Figure 3.10 from Chapter 2 that we used to represent the bowl feeder wall, track, and the parts to be oriented. If we include the set of dynamics parameters comprising

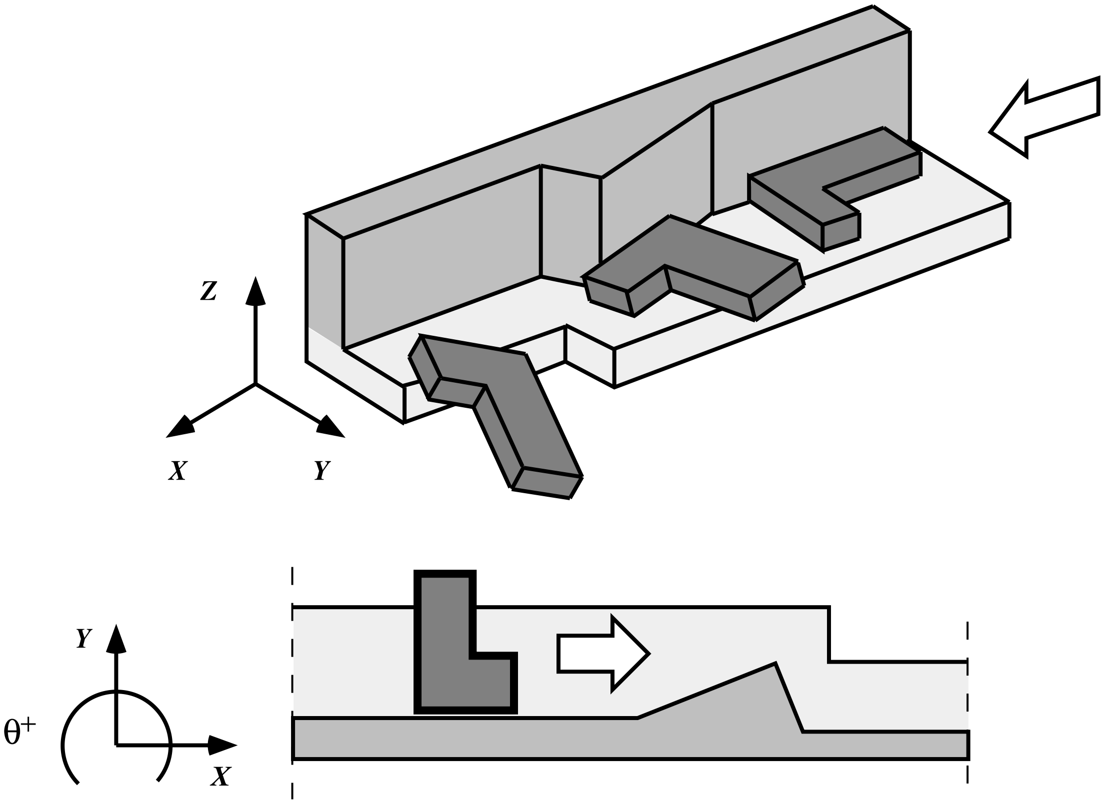


Figure 4.13: A polyhedral model of a portion of the feeder track near the outlet of the bowl (top), and an equivalent planar model viewed from above (bottom).

the applied force and material property parameters, then we have a total of **four** sets of design parameters with which to work.

The degree of coupling between design parameter ↔ motion constraints that we illustrated in Sections 4.1.3 and 4.2.1 is not completely global or all inclusive. Rather, different groups of parameters control and are controlled by different motion constraints that form a rough hierarchy, as illustrated by Figure 4.14. Specifically:

- the kinematic constraints represented by the CS are determined solely by the part and bowl wall interactions,
- the support transition boundaries on the CS are determined by the interaction between the part and track (and also the bowl wall since the support regions are intersected with the CS),[7]

[7]The coupling between changes to the bowl wall and the support transition boundaries is actually

- the forward projections are determined by the dynamics parameters and the CS, including the support regions.

As we further and further constrain the set of possible motions, starting at the top of the figure with the basic kinematic constraints for the CS on down to the non-kinematic constraints that produce the output motions, the motion constraints become more and more heavily coupled to the preceding input parameters. We can visualize each set of parameters metaphorically as an input design "knob" that we can vary to effect changes to the given output. The series of knobs illustrated in Figure 4.14 also offer clues for controlling the complexity of the feeder design task. Specifically, the *lower* the knob in the chain, the lower the degree of coupling between that knob and the other constraints. For example, changing the dynamics parameters (lowest knob) will change the forward projections (output motions) but leave the remaining motion constraints unchanged. On the other hand, changing either the part or bowl wall geometries (top knobs) introduces changes in the motion constraints that propagate through all of the subsequent constraint representations. Thus, the input knobs in Figure 4.14 can be viewed as forming an inverted pyramid of coupling, and hence design complexity.

The bi-directional arrows between the **(Part & Bowl)** $\leftrightarrow$ **CS** and **(CS & Track)** $\leftrightarrow$ **Supported CS** indicate the existence of both forward **and** apparent-inverse mappings available between these parameter & constraint representations, while **(Supported CS & Dynamics)** $\rightarrow$ **Motions** is currently only a forward mapping. Another set of design knobs comes from the non-parametric track cutout function of Section 4.2.2, where we may select the (x, y, θ) position of the cutout as well as the orientation of the fall axis. Because the track cutout function is a non-parametric operation, the hierarchical grouping of the parametric/constraint coupling illustrated in Figure 4.14 does not apply. As a result, invoking the cutout function may affect any and all constraints at any level, and hence makes the control of design complexity more difficult.

4.4.2 Radial Part Symmetry

According to Figure 4.14, the set of parameters describing the part geometry are the most heavily coupled to motion constraint representations of any of the design parameters. This is not surprising since it is the part geometry upon which the entire

an artifact of our decision to represent only the *intersection* between the CS (formed by the part and bowl wall) and the general (x, y, θ) support constraints discussed in Section 2.4.3 and illustrated in Figure 2.12. If we were to represent these support constraints directly as another surface in configuration space, then the CS would be coupled to the part and bowl wall, whereas the support constraint surface would be coupled to the part and the support track. The additional coupling we between the bowl wall and the support constraints we have in Figure 4.14 is one of the prices we pay for a simplified motion constraint representation.

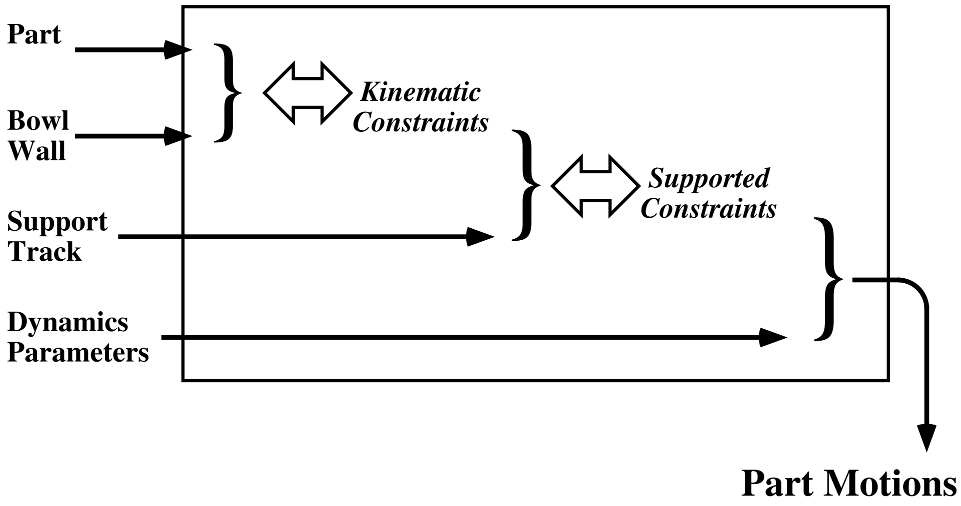


Figure 4.14: Mappings between design parameters and motion constraints viewed as design “knobs”.

feeder design is based. We have stressed all along that it is the *interactions* between shapes, rather than the individual shapes themselves, that determine function. With this in mind, are there any specific properties of a part's geometry we might focus upon initially when considering interactions with a yet to be designed feeder? In particular, we know that before a part reaches and interacts with the portion of the bowl feeder that we will focus our attention on in designing a motion filter (i.e. the final segment of feeder track near the output of the feeder), the parts will be sorted naturally into one of their initial orientations by interactions with a featureless bowl wall (see Section 3.2). In this initial sorting operation, the constraints provided by both the bowl wall and supporting track are important, although in some sense trivial. That is to say, we consider the bowl wall to be locally straight and the track wide enough to support an individual part in any orientation.[8]

Figure 4.15 illustrates the radial symmetry of a part in contact with a straight bowl wall expressed in terms of the distance of the part's reference point from the wall as a function of the orientation of the part. The curved arcs represent contact between a single vertex of the part and the straight wall, while the cusps between the curves represent a contact between an edge of the part and the straight wall. This representation of a part's radial symmetry is also referred to as the *radius function* of the part [34], and is one characterization of the inherent symmetry, or lack thereof,

[8]In terms of the design input knobs of Figure 4.14, the second and third knob have no real effect at this point.

of a planar object. We have seen the radius function a number of times before, although in a slightly different form. Figure 4.16 shows a view of a CS formed by the part in Figure 4.15 interacting with a portion of a straight wall. In Figure 4.16 we are looking along the x axis of the (x, y, θ) configuration space so that we are seeing the y vs. θ profile of the CS. The curved surfaces of the facets and valleys between them **are** the radius function.

From a design point of view the part's symmetry, as characterized by the radius function, represents what we're given to work with. Specifically, the cusps formed by the edge-edge contacts in Figure 4.15, corresponding to the edge-edge valleys on the CS surface, are a superset of the stable initial orientations into which a part will settle as it moves up the track. The relative heights of these cusps in the radius function are a measure of the *separability* of the various stable part orientations. A part whose cusps are all at nearly the same height (i.e. distance of the **cg** from the bowl wall) is nearly symmetric, and therefore its stable orientations are nearly indistinguishable from one another as measured from the wall.[9] A part with at least one cusp easily distinguishable from the others, on the other hand, is more likely to be sorted by, for example, a feeder with a simple narrowed track.

4.4.3 The Integration of Part/Feeder Design

The motion constraint representations we have developed for both analysis and design are general enough that they allow modifications to be made to the feeder geometry in the same way as to the geometry of the part being fed. In fact, the tools and methods developed so far do not distinguish between feeder and part. This is significant because, where possible, redesigning a part so as to make it easier to orient can reduce the number and complexity of required feeding devices. In addition, redesigning parts to a new model of an existing product may allow for the reuse of existing tooling and equipment.

Part redesign may also make sense for a number of other, more technical, reasons. First, as we saw in Figure 4.14, the part geometry directly determines each and every motion constraint representation, from the CS to the forward projections. The central role of part geometry in the feeding operation provides a great deal of flexibility in design. In addition, the degree of radial symmetry determined by the part geometry determines the set of initial orientations we will have to filter, as seen in Section 4.4.2, as well as the design strategies to be employed, as we will see in Section 4.4.6.

[9]Consider the extreme case where we are trying to distinguish between stable orientations of a circular disk. The radius function for such a part would be a horizontal line, indicating that **(a)** there were no finite stable orientations characterized by a cusp, and **(b)** the orientations of the part were indistinguishable from one another.

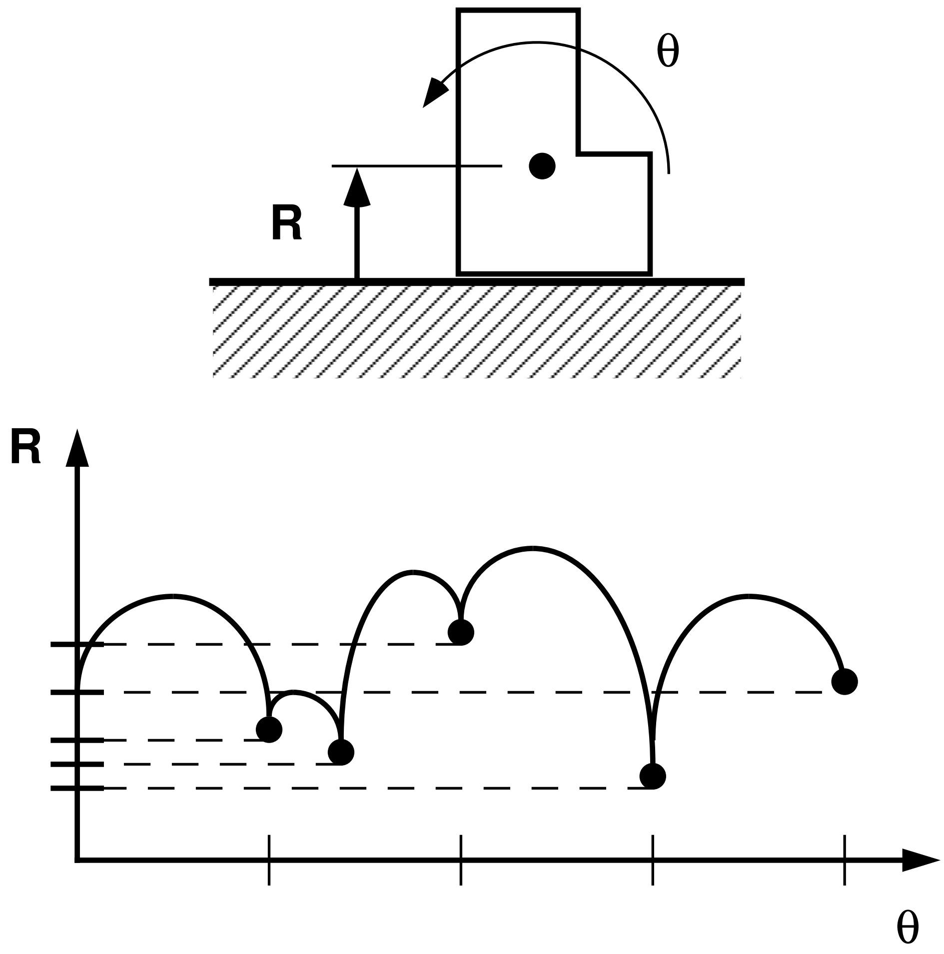


Figure 4.15: The (y, θ) radial symmetry of a planar part interacting with a straight wall.

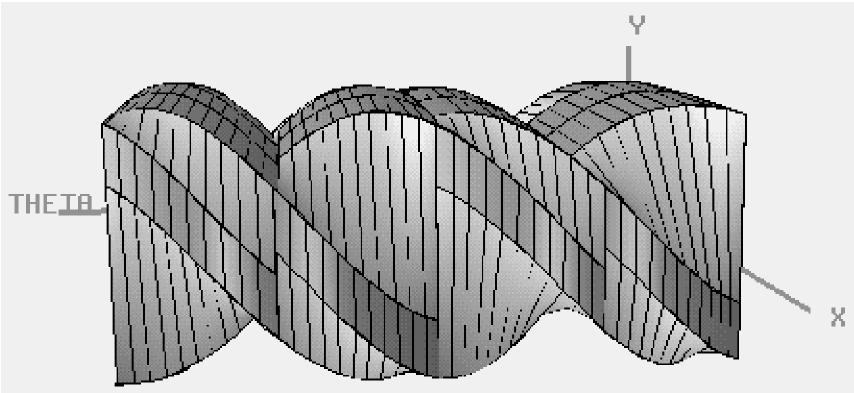


Figure 4.16: The radial symmetry of a part represented by the CS as viewed along the x-axis.

4.4.4 A Coarse Taxonomy of Feeders

Motion constraint representations, such as the CS, provide a detailed and complete characterization of the function embodied in object interactions. Although this detail is necessary both to ensure that a given design will function as intended as well as provide a means of identifying and manipulating critical features, it can be somewhat overwhelming. This is especially true in the early stages of a design task when there is little or no prior experience to guide the search for a suitable design. Ideally, we would like to be able to quickly and approximately determine what classes of object geometries, viewed as regions of a design space, are likely to contain promising designs. Specifically, before generating any detailed motion constraint representations, we wish to select an initial feeder geometry from a qualitative taxonomy of feeder geometry classes. Even a coarse taxonomy for the classes of feeder geometries can be useful in a number of ways in the design of a feeder. In particular, we can use the taxonomy to:

- Aid in selecting a rough initial set of geometries for feeder components that are likely to be "close" to a suitable feeder design in terms of an initial position in the space of design parameters.

- Indicate what sets of parameters (design knobs) should be varied to achieve the desired functional properties (i.e. suggest a rough direction to search in design space).

- Provide the basis for a qualitative mapping between classes of feeder shapes and motion constraint features in configuration space.

	Simple Profile	**Non-simple Profile**
Bowl Wall Polygon	*Straight Wall*	*Shaped Wall*
Track Polygon	*Straight Track*	*Shaped Track*

Table 4.1: A coarse taxonomy of feeder component shapes.

Complexity	Profile Combinations	Design Variables
1	Straight Track & Bowl Wall	Track Width
2	Straight Track & Shaped Bowl Wall	Wall & Dynamics
3	Shaped Track & Bowl Wall	Track
4	Shaped Track & Bowl Wall	Track & Wall & Dynamics

Table 4.2: Possible combinations from the feeder taxonomy.

Table 4.1 illustrates an extremely coarse taxonomy for the shapes of the track and bowl wall. The distinction between a *simple* profile and a *non-simple* profile in Table 4.1 is based on the size of the feeder feature as compared to the size of the part. Roughly speaking, a feeder feature (such as an edge or series of edges) that is larger than *twice* the size of a part may be considered a simple feature because it is locally equivalent to a straight edge. What we are looking for from *non-simple* bowl wall features are contact facets on the surface of the CS that are formed by interactions with part features that are not accessible by a straight wall. For the support track, we are looking for support transition features that, through local interactions with the part geometry, produce support regions that are distinctly different across the surface of the CS in the θ dimension (i.e. distinct support regions for different orientations of the part). The interpretation of this distinction is admittedly rather vague. For example, for two *simple* long edges meeting at a sharp angle the vertex and local edge segments could be considered a *non-simple* feature.[10]

Table 4.2 gives an enumeration of the four possible combinations, or feeder classes, of entries from Table 4.1, along with their relative design complexity and the design parameter sets (knobs) that must be tweaked to obtain a design within the given class. The complexity serves both as an index to a class as well as a qualitative ranking of the expected difficulty of designing a given feeder class as measured by the number of design variables and the anticipated degree of parameter-constraint coupling. Despite the coarseness of Table 4.1, the classes in Table 4.2 highlight a number of interesting and useful qualitative properties of feeder design. For example, in both classes *1* and *3*, the simplicity of the straight bowl wall essentially renders the dynamics parameters useless – the parts will remain constrained within their stable

[10]We qualify this vagueness by noting that a coarse taxonomy, even if it provides initial geometries that turn out to be distant from the final design, are useful for giving us a place to start our search for a better design.

	Simple Profile	**Non-simple Profiles**	
Bowl Wall	*(a) Straight*	*(b) Protrusion*	*(c) Cavity*
Track	*(d) Straight*	*(e) Protrusion*	*(f) Cavity*

Table 4.3: A slightly refined taxonomy of feeder component shapes.

orientations as they pass along the wall. The only design variables we have available to us in these feeder classes are those defining the track profile (class *3*) and its location relative to the bowl wall (class *1*). In terms of motion constraints, we must rely solely on modifications to the support regions on the CS to filter part motions. In (class *1*) in particular, the success or failure of a feeder design will be determined by varying one parameter specifying the track-bowl wall offset. The success or failure of this class of feeder will be determined by the differentiability of part orientations inherent to the part geometry, as given by the radius function in Section 4.16. Class *3* allows us to utilize detailed part geometry for the purposes of differentiating motion paths only by means of varying the support interactions between the part and track contours. From the motion constraint representations this means that, as for class *(1)*, we can only manipulate the support transition boundaries (albeit more complex and localizable ones) on the CS surface.

Classes *2* and *4* from Table 4.2 involve interactions between a part and *non-simple* bowl wall features that may allow for the differentiation of motion paths beyond what is inherently given by the part geometry. Specifically, for class *(2)* we may manipulate paths on the CS by changing *(i)* the dynamics, and *(ii)* the bowl wall profile to redirect the individual motion paths of the parts.[11] In class *(4)* we have the combined effects, and therefore complexity due to coupling, of all sets of design variables. Finally, we note that both classes *(3)* and *(4)*, in which the track profile is *non-simple*, are judged more complex than *simple* or *non-simple* bowl wall profiles because of the highly coupled and non-linear nature of the support transition boundaries as noted in Section 4.2.1.

The feeder taxonomy illustrated in Table 4.1, and the feeder classes derived from it in Table 4.2, provide us with a reasonably coarse, first-cut grouping of the basic feeder feature types we were looking for to begin the design process. One problem with using these tables in their present form, however, is that they do not directly tie in to any particular classification of the motion constraint features in configuration space that we will have to use in our search for more detailed designs. To address this shortcoming, we will generate a *slightly* more detailed taxonomy of feeder features shown in Table 4.3. The basic difference between Tables 4.1 and 4.3 is a minor refinement of the term *non-simple* into two sub-classes denoting protrusions from

[11] Unfortunately, in the present representation, changes to the CS will also produce changes in the supported regions on the surface of the CS. See Section 4.4.1.

and cavities in either a track or bowl wall profile.[12] The important difference here is that elements from this refined taxonomy may be mapped into the set of qualitatively distinct motion constraint features in configuration space given below. We use letter indices to the elements of Table 4.3 to avoid confusion with the feeder classes of Table 4.2.[13]

(a) The straight bowl wall produces a series of parallel valleys running perpendicular to the θ axis of the (x, y, θ) configuration space. These valleys form a "washboard"-like surface made up primarily of type B contact facets, which act to constrain and guide the naturally differentiated stable orientations of a part along the wall, as shown in Figure 4.17 **(a)**.

(b) A protrusion from the bowl wall produces a ridge across the surface of the CS in the θ direction of the configuration space, shown in Figure 4.17 **(b)**. A ridge is typically made up of both type A and type B facets, and serves to differentiate motions further than is possible with the valleys in **(a)** because many parts of a given path are across a single contact facet and therefore may be manipulated by adjustments to the dynamics parameters as well as to the facets themselves.

(c) A concavity in the bowl wall produces a valley across the CS surface in the θ direction as shown in Figure 4.17 **(c)**. This valley, also consisting generally of both type A and type B facets also acts to differentiate motion paths, or in some cases may be used to *combine* motion paths (i.e. part reorienting).

(d) A straight track profile (within the half-width of a part from the bowl wall) will produce supported regions, or *passes*, along deeper valleys on the CS surface, as shown in Figure 4.18 **(d)**. If the straight section of the track is not parallel to the straight portion of the wall, the passes will become wider or narrower along the x direction of a CS valley.‡

(e) A protrusion from the track profile may produce isolated islands within unsupported regions or entire swaths of support on the CS in the θ direction of configuration space, as shown in Figure 4.18 **(e)**. Both isolated islands and supported swaths are rather useless alone since they are not reachable by any paths starting outside the unsupported regions. For this reason, track protrusions are typically combined with bowl wall protrusions to form passes across ridges on the CS so that some differentiated paths are supported whereas others are unsupported as the parts rotate and translate across the track surface.‡

[12] Again, we distinguish between *simple* and *non-simple* features based on the relative size of the feature to a part.

[13] We will not bother to enumerate the *nine* possible combinations of the bowl wall and track elements from Table 4.3.

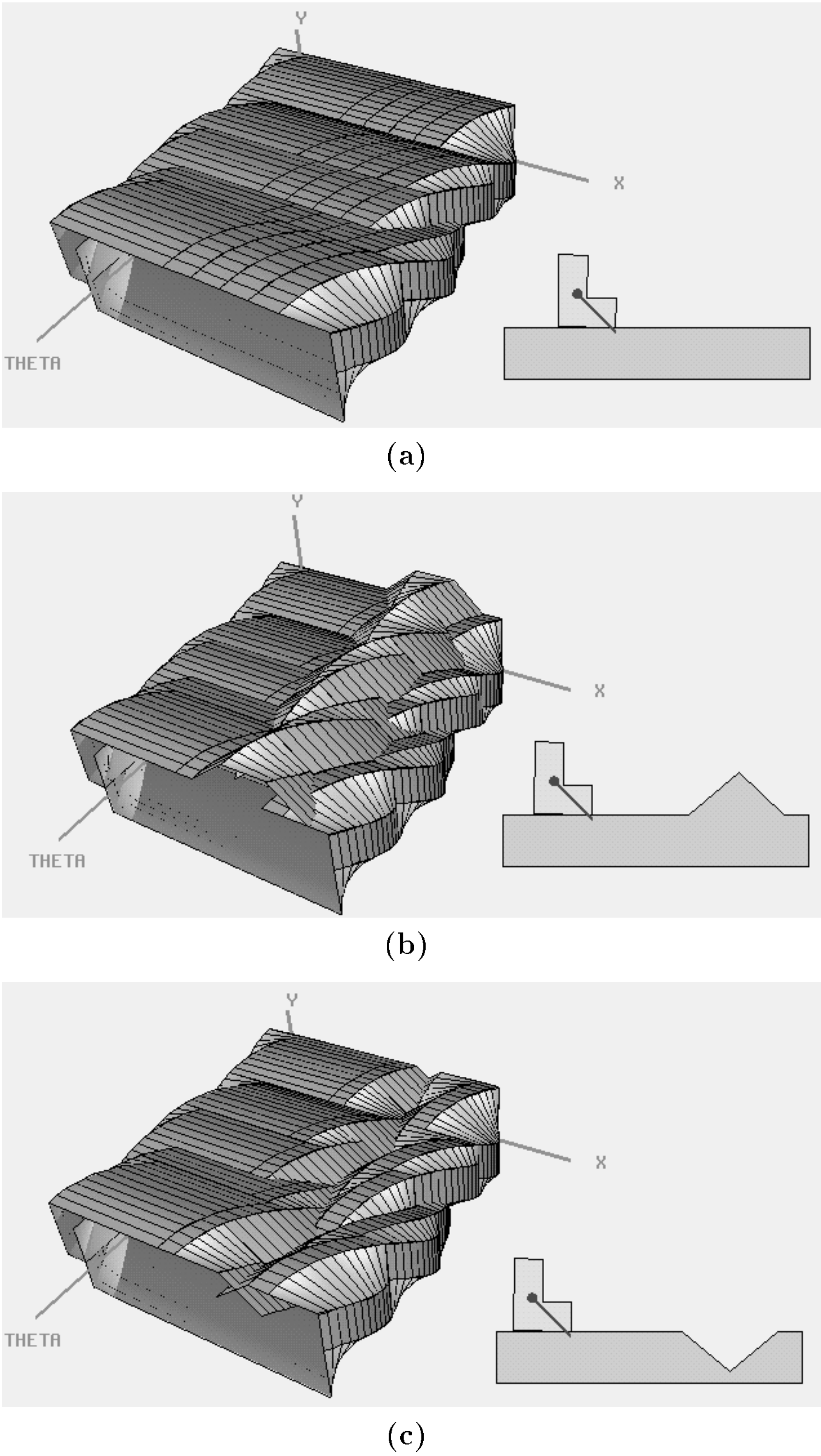
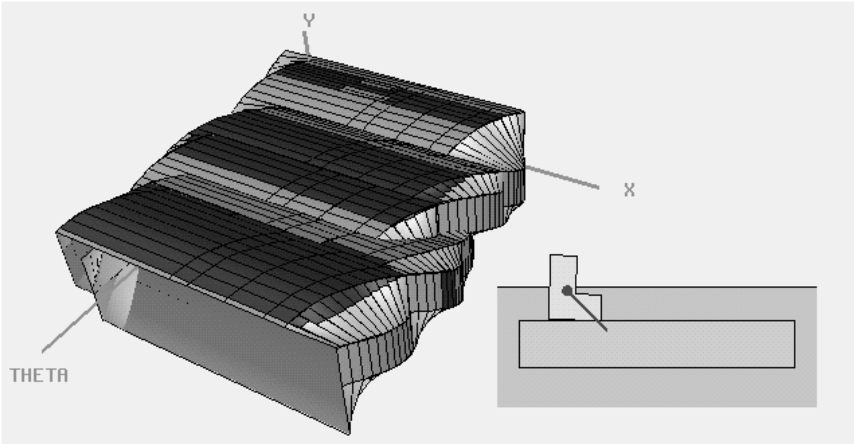
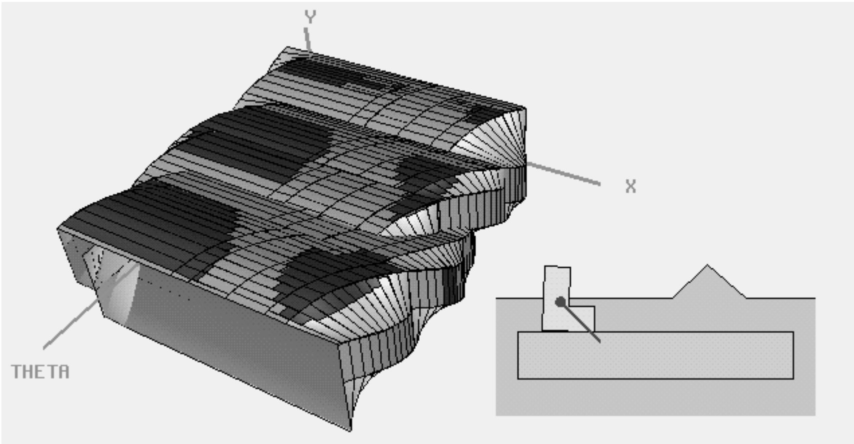


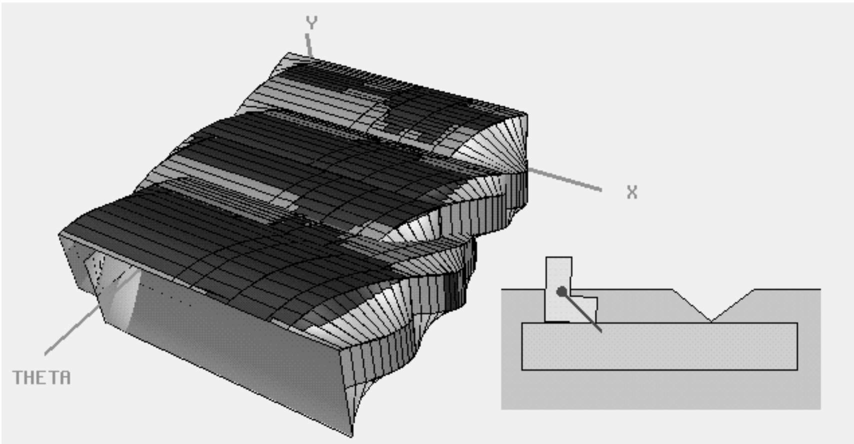
Figure 4.17: A taxonomy of CS features in (x, y, θ) configuration space for different classes of bowl wall profiles.



(d)



(e)



(f)

Figure 4.18: A taxonomy of motion constraint features in (x, y, θ) configuration space for different classes of support track profiles.

(f) A concavity in the track profile will produce unsupported regions across the CS that may appear as isolated islands or entire swaths along the θ direction, as shown in Figure 4.18 **(f)**. Unlike the supported regions in case **(e)**, however, these unsupported regions may be very useful by themselves since they may be used to intercept and terminate selected motion paths. The track profile produced by the cutout operation of Section 4.2.2 produces such features.‡

‡ The support regions generated on the CS for cases *(d) - (f)* assume a simple CS surface, as in case *(a)*.

We could, of course, generate even finer taxonomies of feeder features than those given in either Table 4.1 or Table 4.3. However, there is a tradeoff between the information gained from a finer classification and both the increased size and added complexity of indexing and interpreting the combinations of feature types. Detailed taxonomies have in fact been developed for more general types of feeder than are treated here (see Boothroyd et. al. [8]). The disadvantage of such classification schemes quickly becomes apparent when we observe minor changes made to a feeder geometry within a given class producing major changes in feeder behavior, making it necessary to refine even further the classifications forming the taxonomy. Furthermore, the notion of similarity among classifications is not easily represented in terms of proximity within such a taxonomy – similar classes could be placed in different regions whereas markedly different classes could become closely grouped. These were among the primary motivations behind our choice of motion constraints vs. raw geometry as a representation of function. Once again we stress that the primary role of the taxonomies described here is to identify promising feeder classes with which to *start* the detailed design process that we will describe in the following section.

4.4.5 Design Strategies

Among the key aspects of an interactive design process are: *(i)* getting started with a nominal design, *(ii)* identifying and following promising directions in design space, *(iii)* organizing the search for suitable designs, and *(iv)* limiting complexity to a manageable level. In terms of the concepts presented previously, the taxonomy of feeder geometries described in Section 4.4.4 is intended to provide us with a simple set of suitable initial feeder designs, while the motion constraint representations from Section 3.2 and the design functions from Section 4.2 allow us to evaluate and modify feeder designs and observe the effects of those modifications dynamically (Section 4.1.3). The last two aspects, organizing our design activities and controlling complexity, have been partially addressed in Section 4.4.1 describing the effects of different "design knobs" on the relationship between various design parameters and motion constraint representations. What remains is to integrate the representations and tools into a *methodology* for design.

In Section 3.2 we illustrated a functional description for a vibratory bowl feeder as a filter on the motions of parts (see Figure 3.9). We identified two primary classes of interactions: *(i)* reorienting of part motions by causing parts traveling in one orientation to reorient themselves (i.e. cause one path to transition or merge with another path), and *(ii)* selective removal of support from part motions by allowing parts traveling in a desired orientation to continue through the feeder while forcing those in other orientations to fall off the support track. The motion constraint representations described in Section 3.2 can capture both classes of filtering interactions, although we will focus most of our attention on the latter as it best characterizes the majority of bowl feeder types, and is in general easier to achieve in practice.

The basic strategy used for designing feeders is to first *differentiate* undesired part orientations from the desired orientation as the parts pass through the feeder by means of the interaction between the parts and the feeder bowl wall. Then, *filter out* those parts moving in the undesired orientations by means of their interaction with the support track. In terms of motion constraints, part motion paths are redirected by modifications made to both the CS and dynamics parameters, and selected paths are made to terminate by intercepting them with unsupported regions manipulated on the surface of the CS.

Given a part geometry, we approach the task of designing a feeder in three phases:

- **Initial problem formulation** – getting started. In this phase we essentially "rough out" an initial design starting with the simplest appropriate feeder geometry, typically a class 1 narrowed track feeder from Table 4.2 consisting of a straight bowl wall and support track. From the motion constraints generated by this choice we select the desired part orientation that we wish the feeder to *accept*, with the remaining orientations to be rejected.

- **Constraint manipulation** – the heart of the design process. With the initial design problem defined above, we begin exploring the surrounding region of design space.[14] We first examine the inherent differentiability of the stable part orientations with a narrowed track. If we are able to obtain only the desired orientation by this method, i.e. if the part geometry is naturally orientable into the desired orientation, then we are done. Otherwise, we begin the detailed feeder design process, with the goal of reducing complexity as much as possible by varying only a few parameters at time. We do this by alternatively focusing on apparent inversion of either the CS $\leftrightarrow$ bowl wall or of the support transition boundaries $\leftrightarrow$ support track. We explore a wide range of variations to a selected set of parameters before moving on to another set since, due to the nonlinear behavior of the constraints, larger variations may have characteristically different effects than small ones. We use the non-parametric cutout

[14]In exploring a local area in design space we're basically assuming that the space is locally smooth and continuous, as noted in Section 4.1.4.

operation sparingly, if at all, and only early on in the design process. At all times we try to avoid terminating the desired path, and immediately restore it if we should inadvertently do so. Finally, we verify the behavior of the design continually as it evolves by observing the representations of the motion constraints, and by occasionally animating the forward projected motion paths.

- **Problem redefinition** – figuring out what to do when you get stuck. There is no guarantee that we will be able to find a feeder design suitable for the given formulation of the problem, or that such a design even exists. We have two ways of redefining the problem should we fail to make progress toward a solution. The first is to simply choose a different orientation that we wish to accept, and begin the constraint manipulation process anew.[15] The second approach is to subdivide the design problem by taking the output part orientations from whatever feeder design we obtained above and treat them as the input to a new feeder design problem.

Figure 4.19 illustrates a feeder design methodology, based on the above phases, in the form of a flowchart. The functional blocks embedded within each of the phases are described in detail below.

1. **Select initial feeder class:** Begin with the simplest class of feeder (class 1 from Table 4.2 consisting of a straight bowl wall and track).

2. **Generate motion constraints:** Construct the CS, support regions and motion paths using the nominal feeder geometries and dynamics settings.

3. **Select desired path:** Choose one path to pass through the feeder without losing support. To maximize feeder throughput, it is best to select the path corresponding to the part initial orientation with the highest probability, illustrated in `cspace-shell` by the *thickest* path.

4. **Try a narrowed track:** Try to exploit the natural differentiation of part orientations (radius function) by varying the offset of the straight track edge from the straight bowl wall (class 1) so that only the desired path passes through the feeder.

5. **Remove other paths:** If the remaining paths cannot all be terminated by unsupported regions while at the same time maintaining support for the desired path, we choose the next *thickest* unterminated path and:[16]

[15] In some cases, the result of the constraint manipulation phase may be a feeder that accepts the wrong part orientation, but rejects all of the other orientations, including the desired orientation. In this case, we may simply choose to accept the result and declare the problem solved.

[16] The majority of design activity will occur in step 5 between **(ii)** and **(iii)**. We must continually monitor the desired path's status and stop modifications short of terminating that path. If the desired path *does* become terminated, we should first attempt to restore it before continuing.

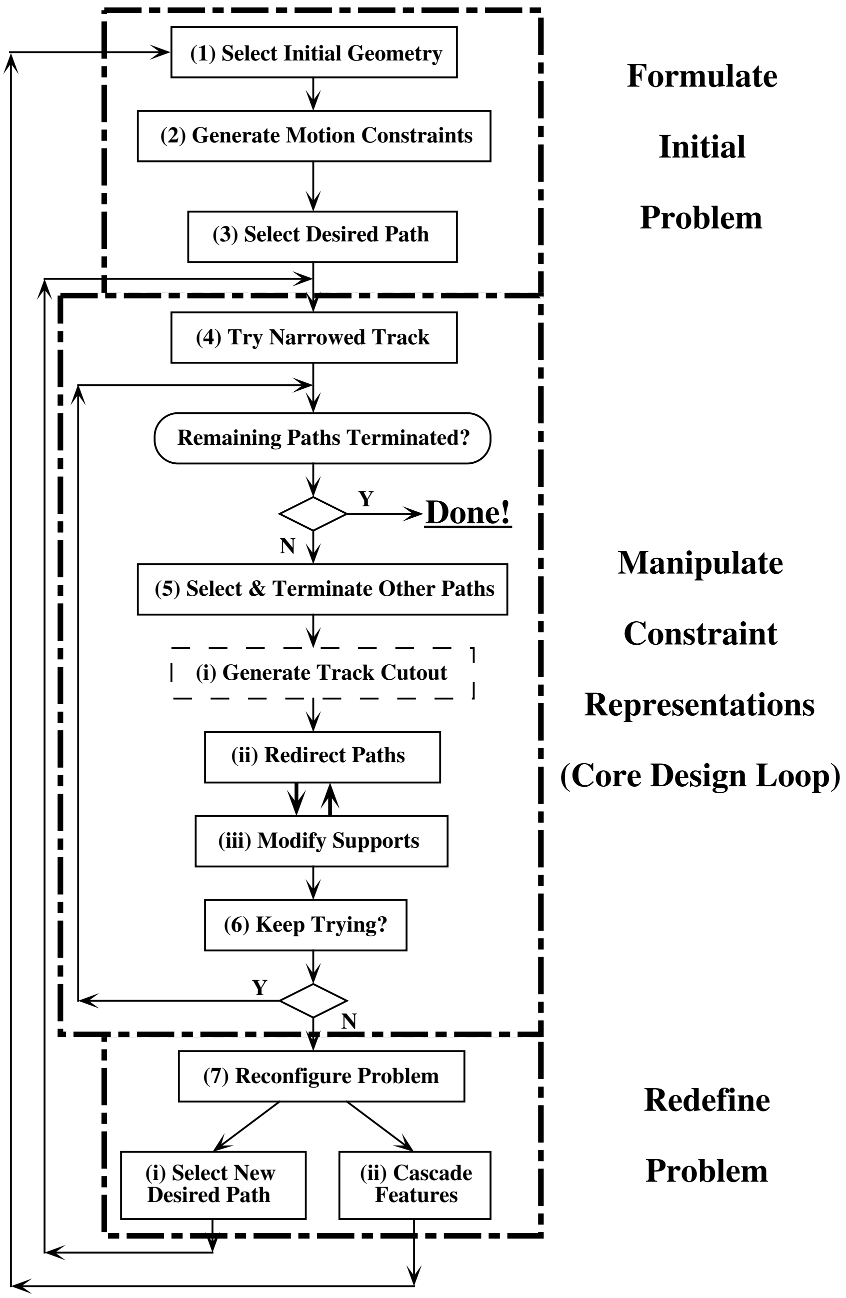


Figure 4.19: Flowchart for a bowl feeder design strategy (see text).

(i) **Track cutout:** Early on in the design process, we may try introducing a **cutout** near the unterminated path with the most differentiated valley height from the desired path. (If the unterminated path is *higher* in y than the desired path (most likely case) then choose a **left** falling cutout near the unterminated path. Otherwise, choose a *right* or *forward* falling path near the unterminated path.) This corresponds to a class 3 feeder geometry from Table 4.2.

(ii) **Redirect paths:** Manipulate the CS surface to differentiate the unterminated path from the desired path by creating or modifying a ridge or valley on the CS (i.e. a bowl wall protrusion or cavity, features **(b)** & **(c)** from Table 4.3). This corresponds to a transition from a feeder class 1 $\rightarrow$ class 2 transition from Table 4.2. For non-class 1 bowl wall geometries, we may also vary the applied force vector to further differentiate paths across individual contact facets.

(iii) **Modify unsupported CS regions:** To *intercept* the unterminated path, manipulate the support transition boundaries on the CS near a portion of the unterminated path on the CS ridge or valley feature. This corresponds to a transition from a feeder class 2 $\rightarrow$ class 4 transition from Table 4.2.

6. **Repeat for other unterminated paths:** Select other unterminated paths and repeat step 5 until only the desired path passes through the feeder.

7. **Reconfigure the design problem:** If no progress is made after trying *a number* of other paths:[17]

(i) **Select another desired path:** One option is to choose another path as the desired path. Specifically, we choose the next *thickest* path as the desired path and go back to step 4. In doing so, we are basically we're accepting a reduced feeder throughput.[18].

– **OR** –

(ii) **Cascade filters:** If some paths have been removed by the feeder features designed so far, we can locally decouple the problem by *cascading* feeder features. Specifically, we take the remaining paths from the existing set of feeder features and go back to step 1. Essentially, we are designing a new feeder for the remaining paths inherited from the previous feeder in the chain.

[17] The term *a number* is intentionally vague. Depending on the particular example, the information provided by dynamic visualization of constraint coupling should quickly tell us whether or not a particular local manipulation strategy will pay off.

[18] Before doing this one might want to save the current feeder geometry and start out with new nominal bowl wall and track profiles (i.e. class 1 from Table 4.2)

The traversal of the steps illustrated in the flowchart is carried out in three **nested** loops, each of which deals more locally with a sub-problem of the design task and in more detail than the enclosing loops. The three loops, each of which may be iterated on a number of times in the course of a design, are given as:

- **Feeder feature groupings** – localization and cascading of feeder features.
  - **Selection of desired path** – formulate the *accept* (i.e. passing) motion goal so as to optimize the feeder throughput.
    - **Direct manipulation of motion constraints** – modify the bowl wall and track polygons by manipulating the CS and support transition boundaries, respectively, while focusing on each individual motion path.

The innermost loop, forming the core of the design methodology – constraint visualization and manipulation – is embedded in the representations and design functions that form the `cspace-shell` environment. In Figure 4.19 this corresponds to steps 5(i), (ii), (iii) and step 6. Surrounding this loop is the selection of the desired *accept* path corresponding to step 3, in the initial problem formulation phase, and to step 7(i) in the problem redefinition phase of design. Finally, the outermost loop involves selecting an initial (simple class 1) feeder geometry that generates a series of motion paths for a given part geometry (step 1). During the problem redefinition phase this involves chaining the output of a partial feeder design into a subsequent feeder design problem (step 7(ii)).

4.4.6 Feeder Examples

Figure 4.19, is intended to serve as a roadmap for using the representations and design tools in `cspace-shell` more effectively by providing a consistent and organized means of searching the space of designs for a solution while at the same time controlling the complexity inherent in the design process. For a given example, the intuition and experience gained by the designer might suggest alternative strategies based on the appearance of certain motion constraint features or constraint coupling characteristics observed during design. Indeed, the purpose of the representations and interactive tools is to provide the necessary flexibility to the human designer to deal with *individual cases* in a correct and consistent manner. In the ideal case, the "right thing" to do during a given design session will become clear from the motion constraint representations as the designer explores the design space.

Example 1

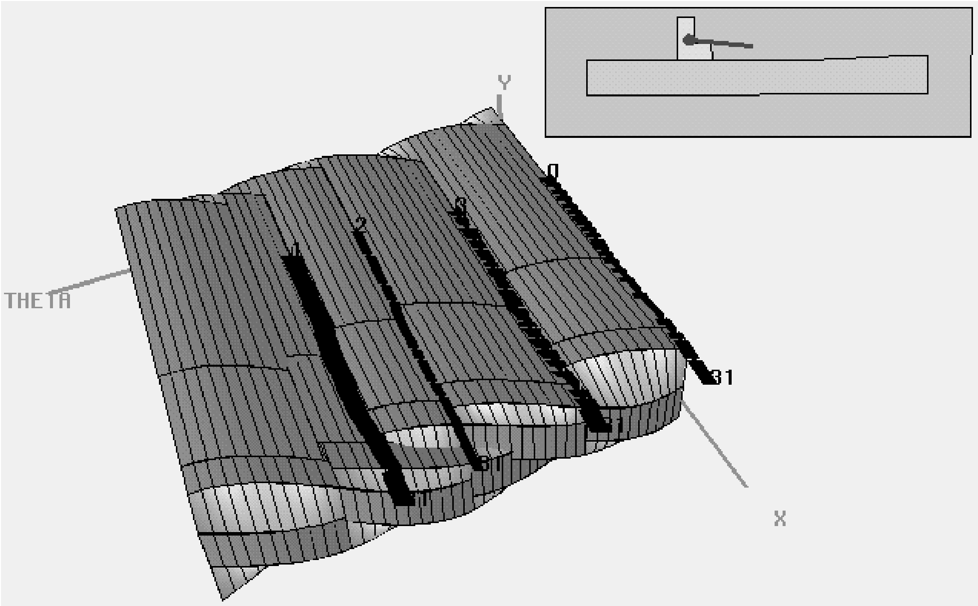


Figure 4.20: Initial feeder geometry and motion constraints for an "L"-shaped part. Note that parts may pass through the feeder in many orientations, as indicated by the multiple paths.

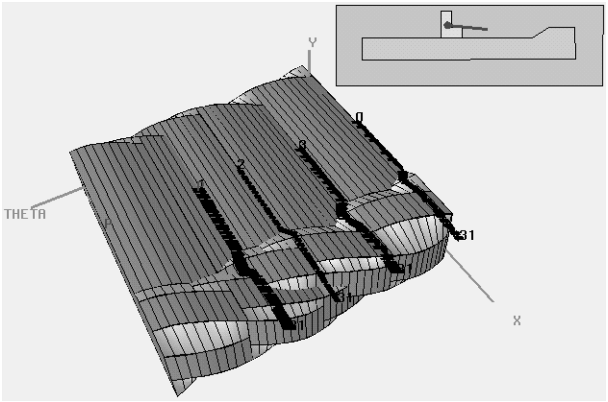
Figure 4.20 shows an "L"-shaped part and an initial nominal feeder geometry consisting of a straight track and a straight bowl wall. The resulting feeder design is shown in Figure 4.22. A few of the steps in the process of generating this feeder design are shown in Figure 4.21, and consist of the following:

(a) Distinguish paths by introducing a bowl wall protrusion. This is accomplished by grabbing a portion of the CS surface between two of the paths and pulling it to form a ridge.

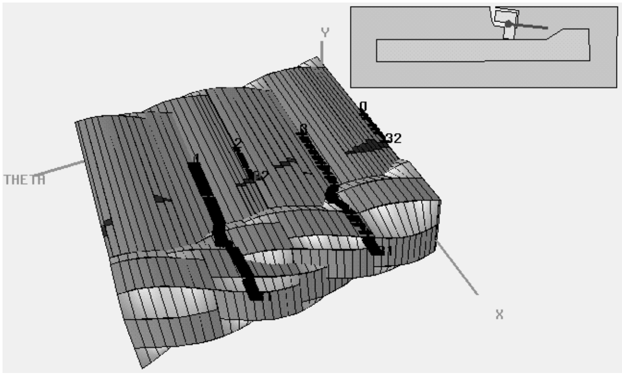
(b) Introduce a notch with the *cutout* operation. The resulting local unsupported region (shaded) terminates the selected path at the desired point, as well as introduces a number of other unsupported regions on the CS surface.

(c) Manipulate the track notch via apparent inverse manipulation of the unsupported regions on the CS. This involves selecting and moving boundaries of the unsupported regions, paying careful attention to the inherent coupling among the boundaries. This process is repeated until the desired path is unobstructed and the remaining paths are all terminated, as shown in Figure 4.22.

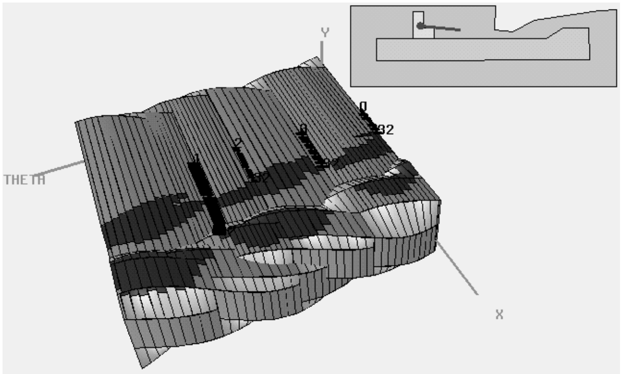
Example 2



(a)



(b)



(c)

Figure 4.21: Intermediate steps in the design of a feeder for the "L"-shaped part in Figure 4.20.

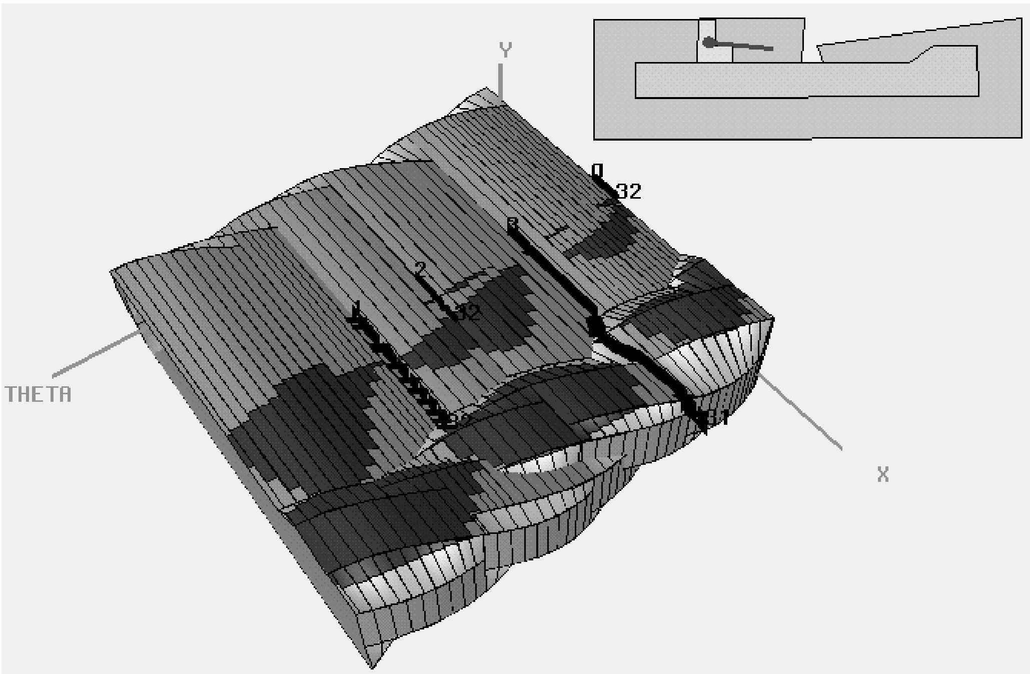


Figure 4.22: Resulting feeder design and motion constraints generated from the initial geometry in Figure 4.20. Note that, as required, parts may pass through and exit the feeder in only one orientation.

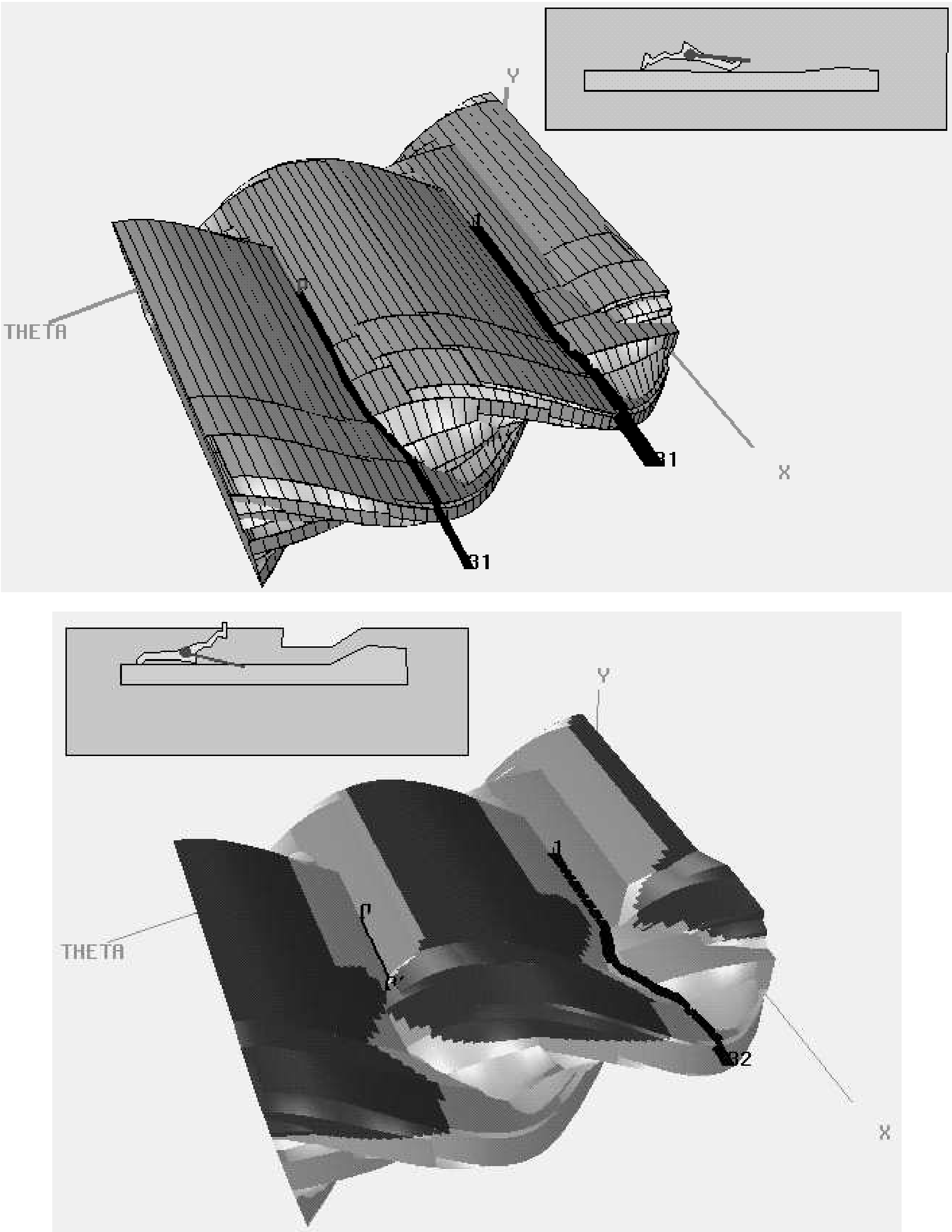


Figure 4.23: Initial (top) and final (bottom) feeder geometries and motion constraints for orienting a plastic cable fastener.

The initial feeder geometry and motion constraints for a part consisting of a plastic cable fastener is shown in the top of Figure 4.23. As in the previous example, a protrusion in the bowl wall was introduced by "pulling" on the surface of the CS to form a ridge in order to help distinguish among the two motion paths, both of which initially travel nearly parallel to one another across the CS in the θ direction. The track contour was developed by initially pulling a pair of points on the track boundary toward the bowl wall to produce unsupported regions on the CS. The unsupported regions were then manipulated within the motion constraint representation via apparent inversion. After a few iterations the motion constraints characteristic of a motion filter were obtained, as shown in the bottom of Figure 4.23, together with the corresponding bowl wall and track geometries.

Example 3

The top of Figure 4.24 shows the initial feeder geometry for an *x-acto* razor blade. Here, as in the previous examples, a protrusion has been introduced into the wall profile to form a ridge on the CS. Unlike the previous examples, however, the purpose of this ridge is to catch and *reorient* those parts that are sliding with their major axis perpendicular to the wall. On the CS, the ridges forming these reorienting constraints may be characterized as fences that are angled relative to the θ direction, thus diverting or guiding multiple motion paths toward a subset of common orientations (θ positions).[19] The rather oddly shaped gap in the track shown in the bottom of Figure 4.24 was introduced using the cutout operator to filter out the undesirable motion paths. The track contour produced by the cutout in this case required only minor modification to produce the final feeder geometry shown.

4.4.7 Physical Experiments

Models for selected part, bowl wall and support track profiles were constructed from plexiglass and tested by placing the parts on the track surface in random initial orientations and allowing them to slide *downhill* while applying a slight vertical vibration to the track surface.[20] Figure 4.25 shows the hardware used to test some of the feeder designs developed using `cspace-shell`.

The part motions, and resulting feeder behavior, closely followed the motion paths predicted by `cspace-shell`. Photographs of one of the experiments run for the part and feeder in Figure 4.23 (top) are shown in Figure 4.26 and Figure 4.27.

[19] As noted earlier, the ability to reorient rather than reject some part orientations has the advantage of increasing the throughput of the resulting feeder.

[20] Since the amplitude of vibration applied to the track was insufficient to cause the parts to "hop", the track was tilted downward so that gravity would drive the part motion.

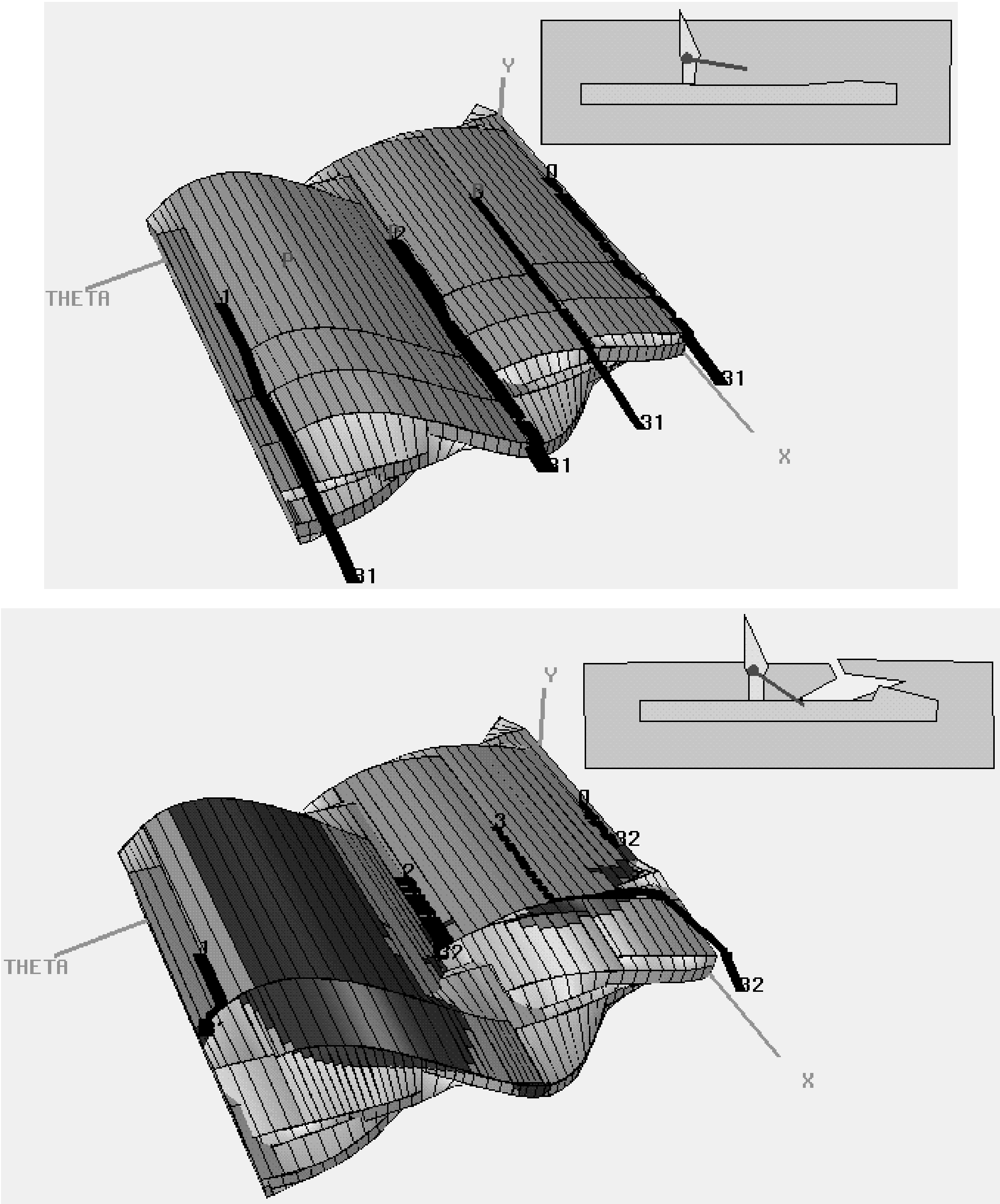


Figure 4.24: Initial (top) and final (bottom) feeder geometries and motion constraints for orienting *x-acto* razor blades.

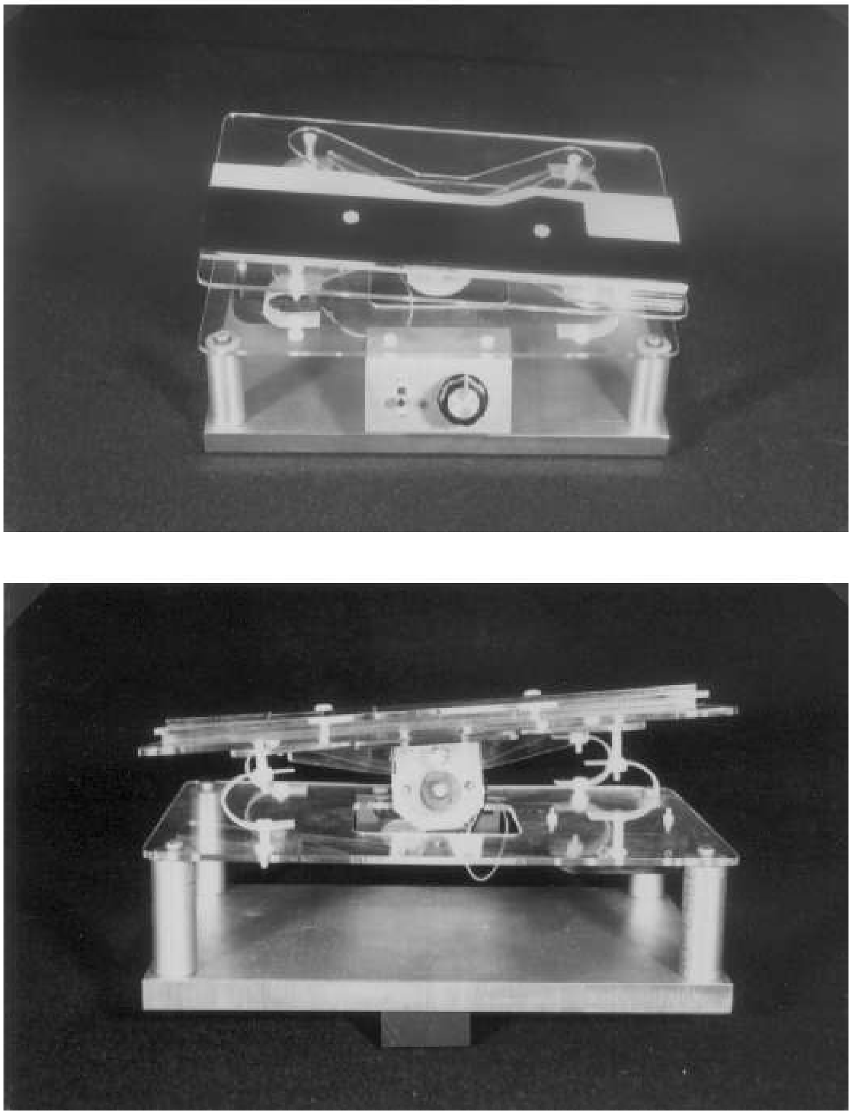


Figure 4.25: Hardware for testing feeder designs consisting of a plexiglass wall and track mounted on a tilted shaker table (top), and a closeup of the shaker assembly (bottom).

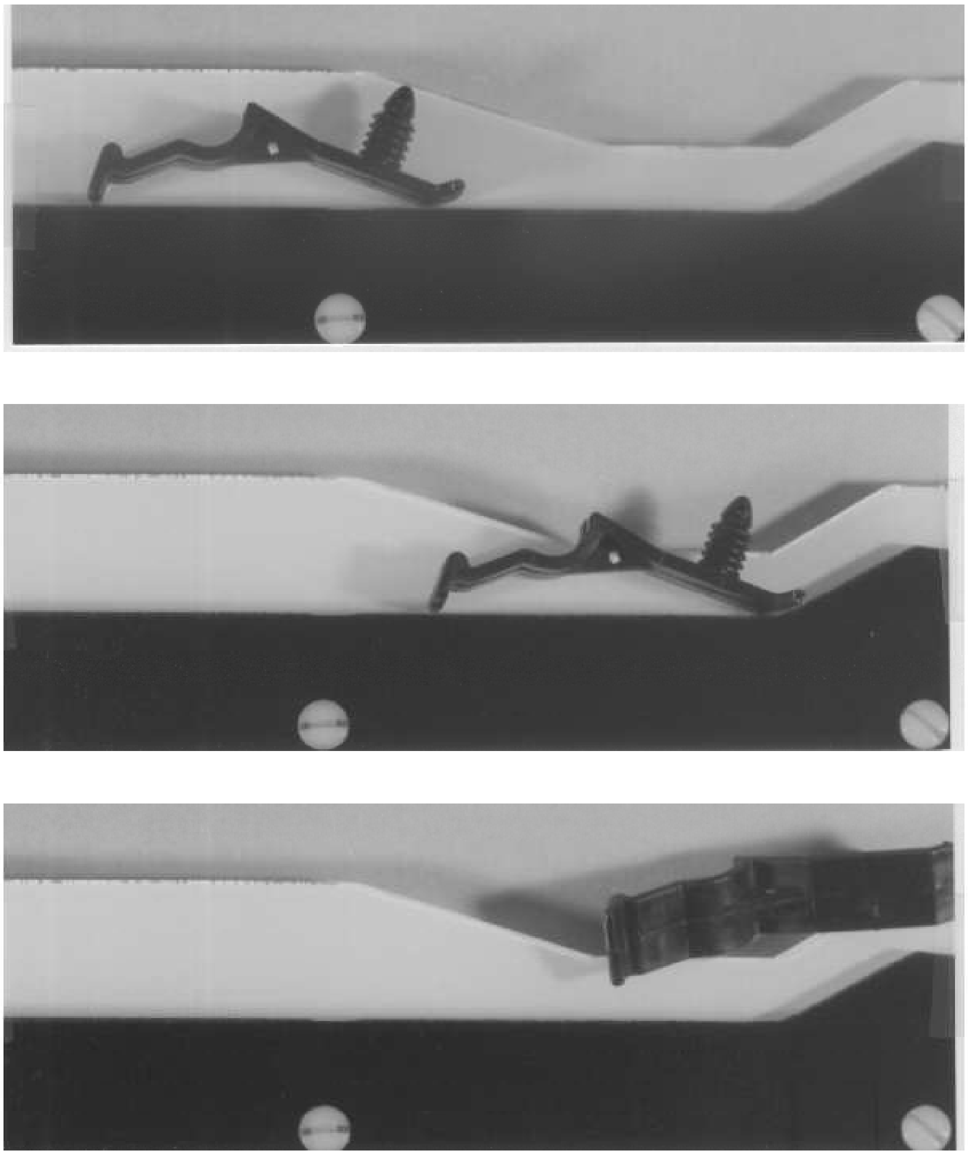


Figure 4.26: Snapshots of a part in the *reject* orientation moving through the feeder design from Figure 4.23.

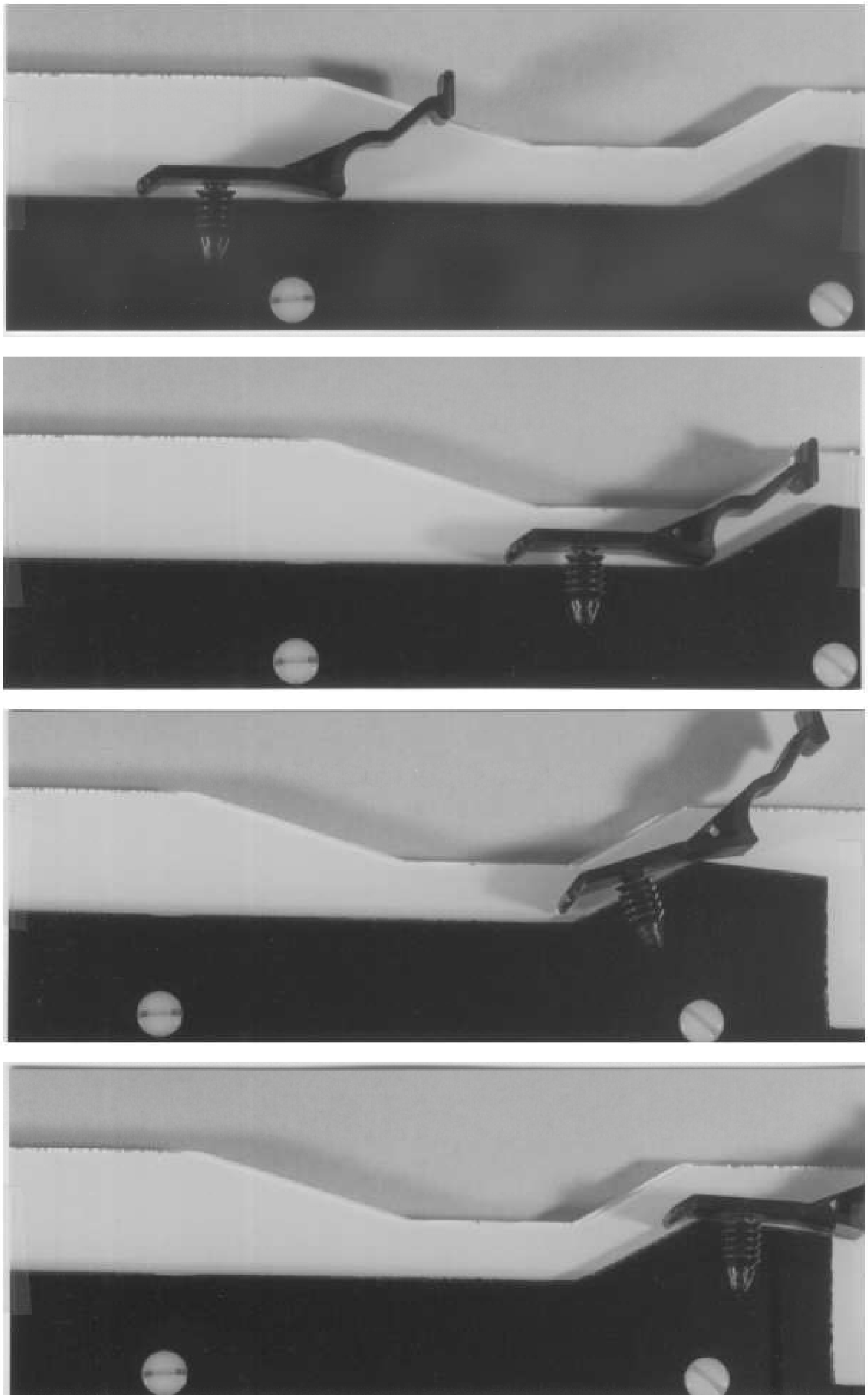


Figure 4.27: Snapshots of a part in the *accept* orientation moving through the feeder design from Figure 4.23.

4.5 The Design of Compliant Assemblies

To further demonstrate the applicability of the representation and manipulation tools developed earlier in this chapter, we will now turn our attention from vibratory bowl feeders to the design and analysis of compliant assemblies introduced in Section 3.1. In particular, we shall illustrate the flexibility of the analysis and design tools developed in `cspace-shell` by adapting them for use in the domain of assembly.

4.5.1 Non-assembly Constraints

The feeders designed earlier have only one functional requirement – orienting parts. Whatever functional requirements that the specific part geometries themselves were required to meet were not considered directly as they were, in a sense, orthogonal to the function of the feeder. In our examples, whatever external constraints on the part geometries that might have existed were implicitly satisfied by the assumption that, in general, the part geometries themselves would not be modified during feeder design.[21] In the case of assembly, however, *both* interacting parts typically have to satisfy a number of constraints that may have nothing to do with the assembly process. Examples of some non-assembly constraints include: maintaining a minimum area if contact between bearing surfaces that must be aligned and unobstructed (e.g. shaft and bearing represented as a peg-in-hole), minimum area contact surfaces for electrical connectors, reference surfaces that must remain exposed to subsequent subassemblies or grippers, part features and dimensions required for strength and stiffness, and features necessary for locating and machining the parts before assembly.

One approach to preserving non-assembly constraints is to simply lock certain object features (both peg *and* hole) into relative configurations at the goal state that ensure the necessary local constraints are maintained. This is equivalent to dividing the set of edges and vertices describing each part into two classes: *mutable* and *fixed.* Modifications to mutable geometric and dynamics parameters that allow these key feature pairs to be brought into contact at the goal state of an assembly operation would also, presumably, satisfy *both* the assembly and non-assembly constraint sets. The advantage of this approach is that non-assembly constraints may be expressed simply as additional constraints on the existing set of design parameters, no matter what their origin. The disadvantage to simply fixing existing parameters is that, since we do not know the origin of the constraints, we have no flexibility in terms of optimizing them together with motion constraints for assembly.

In the following examples we will follow the above approach. For the sake of

[21] For those examples where we did modify the part design, we *assumed* that the designer kept these constraints in mind during the design process so that any modifications would not compromise part functionality.

generality it would be desirable to combine the representations for non-assembly constraints with the motion constraint representations in configuration space wherever possible. Such representations would be particularly useful in the effort to design components and subassemblies *concurrently* in the context of both product and process constraints. Given the wide range of such outside constraints, however, it is likely that different representation combinations would have to be developed for different classes of assemblies. These issues remain an open area for future research.

4.5.2 Methodology

In terms of managing the complexity due to coupling among motion constraint representations, the design methodology for compliant assemblies is considerably simpler than that necessary for vibratory bowl feeders. In particular, since we will focus our attention on the local region surrounding the hole on the CS, as described in Section 3.1, we need not be concerned with the introduction of undesirable constraint features elsewhere on the surface of the CS.[22] In addition, since we do not consider out-of-plane part motions in assembly, we may avoid the very non-linear coupling observed between shape modifications and the support transition boundaries. What remains from `cspace-shell` as developed for bowl feeder design are the kinematic constraints of the CS surface and the forward projections expressed as discrete motion paths. With respect to the feeder design methodology flowchart of Figure 4.19, we may concern ourselves with only step 5(ii) **Redirect Paths** of the innermost loop.[23]

Interactive design of motion constraints for assembly consists of modifying the mutable subsets of both part geometries via apparent inversion of contact facets on the CS surface, varying the reference point of the moving part (i.e. compliance center) directly, and varying the dynamics parameters. We will assume a *generalized damper* model of compliance mapping differences in velocity between the assembly robot and the part into forces on the part (see Section 3.1). Under this model, the force vector applied through the moving part's reference point in `cspace-shell` is interpreted as a commanded velocity from the assembly robot. As for bowl feeders, the dynamics of part motion is assumed to be quasi-static (see Section 2.4.2). For this case, the dynamics parameters consist of the direction of the force/velocity vector applied to the compliance center, which is assumed to be parallel to the intended nominal assembly path, and the coefficient of friction μ.

Since, for the purposes of expediency, we are borrowing design tools originally designed for the design of bowl feeders, a few design operations must be carried out

[22]We should keep in mind, however, that proximity of motion constraint surfaces in configuration space does not necessarily require proximity of geometric features on an object.

[23]The rest of the feeder design methodology is unnecessary for assembly as it is dedicated to dealing with coupling and managing interactions among global constraints.

somewhat indirectly. Changes to the rotational compliance C_θ, relative to the translational compliance terms, may be carried out by varying the normalized effective inertia of the of moving object in the form of the radius of gyration ρ by means of a slider. At present, the apparent inversion tools do not support the manipulation of the moving object's reference point via changes to the motion constraint representations.[24]

The initial position from which each motion path starts, consisting of a (x, y, θ) point in the *free* region of configuration space, is chosen manually via sliders. We explore the behavior of the assembly over a range of parameters possible under positional and control uncertainty by selecting discrete values of the initial position and dynamics parameters, as noted in Section 3.1. The resulting *bundles* of discrete paths sample the full forward projection of motion for the assembly task under uncertainty.

4.5.3 Assembly Examples

Example 1

Figure 4.28 shows the change in motion constraints resulting from the addition of chamfers to both a peg and hole. The constraints shown form the entry region to the hole surrounding the goal region of the assembly (i.e. the peg positioned inside the hole). The chamfers in this example were generated directly from the geometry of the parts, with the motion constraint representation serving only to confirm that the desired functional characteristics had been achieved.

Example 2

Figure 4.29 illustrates another aspect of the design of compliant assemblies. Specifically, the location of the *center of compliance* relative to the tip of the peg determines whether or not the motion constraint boundaries surrounding the goal region of the assembly guide motions to the goal in the presence of rotational misalignments (i.e. offsets in the θ direction), as shown in the bottom of Figure 4.29. The cup-like curvature of the motion constraint boundaries in the θ-direction may be viewed as a form of *rotational chamfer*, whose function is analogous to that of the chamfers introduced to the part geometries in the previous example. Thus, the motion constraint representation serves to illustrate the similar effects achieved by proper placement of the compliant center (assembly strategy) and the addition of chamfers (part geometry) on the success or failure of an assembly task.

[24]It is possible to indirectly modify the reference point from the motion constraints by selecting **all** vertices of the moving polygon from the CS. Uniform modifications to these vertices has a similar effect (but in the opposite xy direction) to modifying the reference point alone.

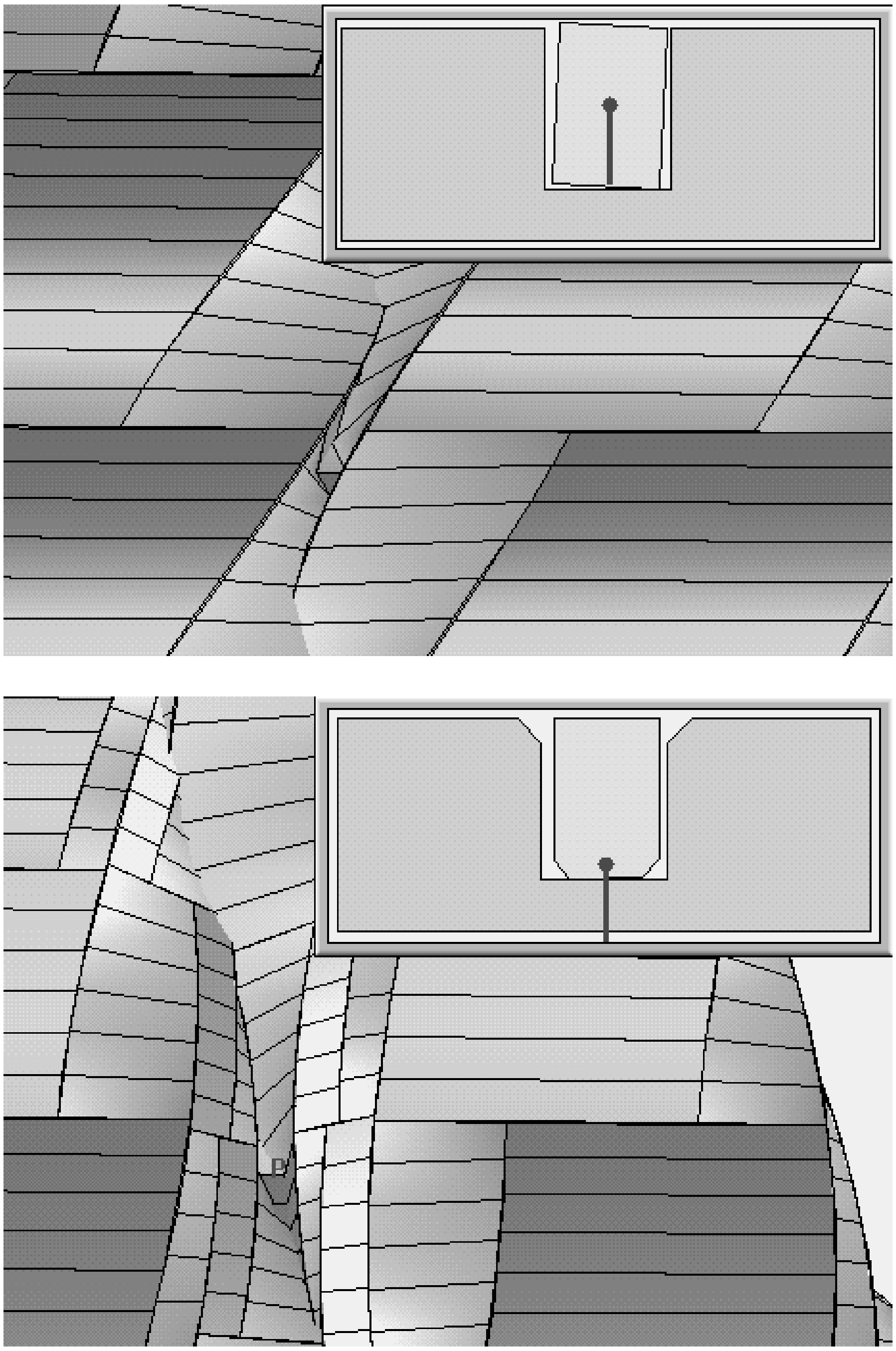


Figure 4.28: Initial (top) and final (bottom) peg-hole geometries and closeups of the motion constraints local to the goal region of a simple compliant assembly task.

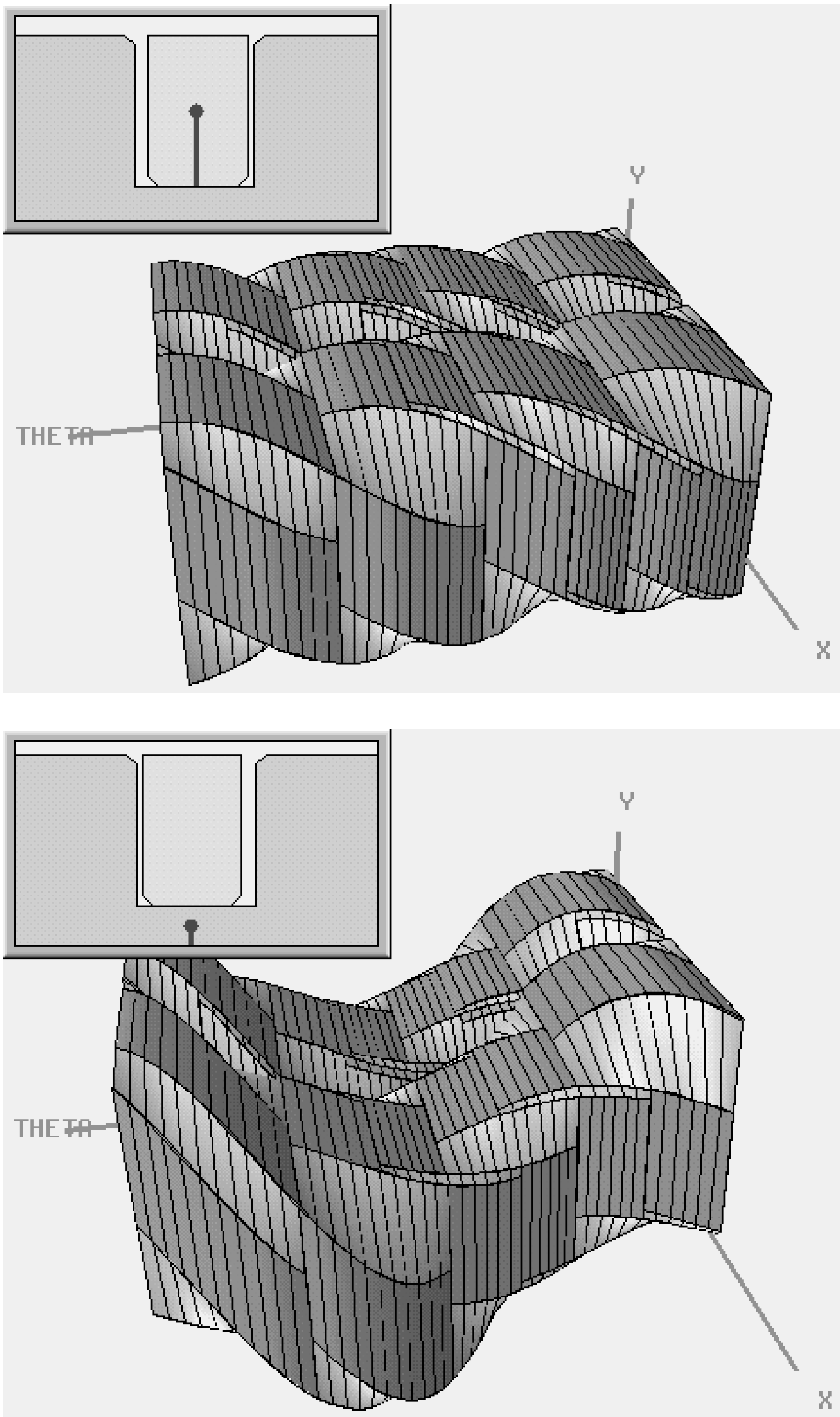


Figure 4.29: The effect of moving the compliant center along a peg's major axis from the middle of the peg (top) to the tip of the peg (bottom), on the rotational aspects of the motion constraints (θ dimension of the configuration space).

Example 3

Figure 4.30 shows a somewhat more complex pair of part geometries comprising an electrical plug connector. Unlike the peg and hole example in Figure 4.28, modifications to the part geometries were made entirely from within the motion constraint representation using apparent inversion. Specifically, constraint surfaces near the goal region in Figure 4.30 (top) caused the assembly to jam in the presence of rotational misalignments. To modify the design, the constraint surfaces in question were selected and pushed out of the way in order to widen the access to the goal region, with the corresponding part geometries being modified accordingly by means of apparent inversion. It is interesting to note that geometric modifications in this example were made to a number of part features (i.e. the entry guides and center pin of the socket) that are not proximal on the part contour but nevertheless produce proximal (intersecting) motion constraint surfaces in configuration space.

4.6 Discussion

We will briefly discuss some of the observations made after using the representations and tools for design. We will also consider some of the limitations of and suggest possible extensions to the implementation of `cspace-shell`.

How useful were the representations and tools for design? As expected, the space of design parameters was *very* large even for relatively simple systems. Given this, the following general conclusions about the use of motion constraint representations for design were reaffirmed:

- The ability to visualize the coupling between parameters and constraints, as well as the sensitivity of a system to local design changes, was crucial in successfully iterating toward a design goal for highly coupled systems like bowl feeders.

- The manipulation of design parameters directly in the context of the motion constraint representations provided a good idea of which way to go (locally) to reach a desired design state, although non-linear coupling often made accurately predicting the effects of large excursions in design space difficult.

- Classifications and taxonomies of shapes were somewhat useful for starting out a design iteration loop by quickly getting to a region of design space. However, the parametric "tweaking" that takes place afterward was a necessary and significant component of achieving a successful design. Basically, taxonomies and shape classifications can sometimes get one *close* to a good design, but almost invariable detailed design is necessary to make things work for a particular example.

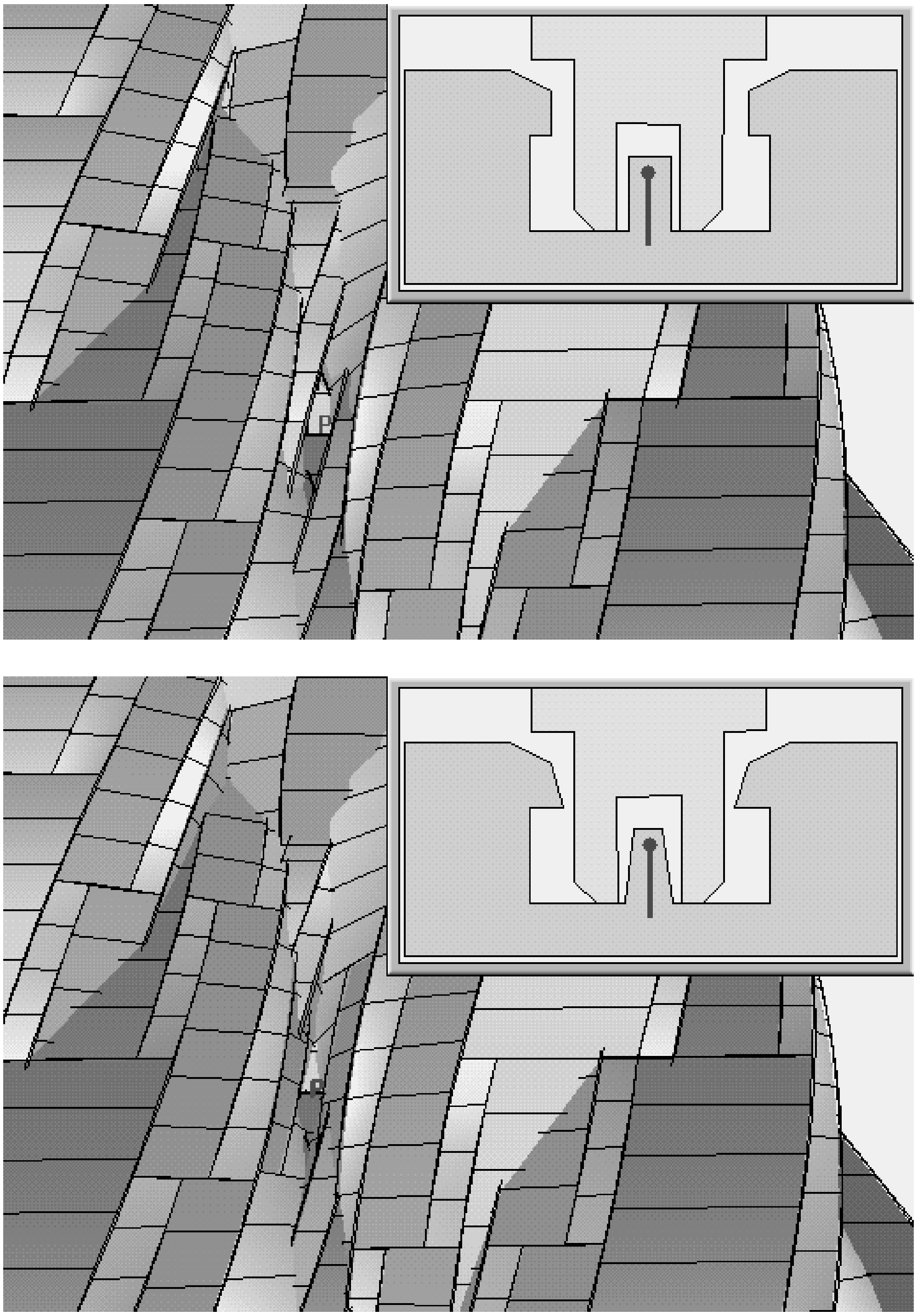


Figure 4.30: Initial (top) and final (bottom) plug-socket geometries and closeups of the motion constraints local to the goal region of a connector insertion task.

How effective and easy to use were the design functions? The inflexibility resulting from a number of the approximations and additional constraints imposed for the purposes of easier implementation of `cspace-shell` proved to be somewhat more of a nuisance than anticipated:

- Generally speaking, the design functions allowed us to reach desired points in design space — eventually.

- For bowl feeder design, having to jump back and forth between modifications to the kinematic constraints of the CS and the support constraints of the support transition boundaries within the innermost design loop of the methodology in Figure 4.19 was awkward. In particular, much of the coupling between constraints at this stage appeared to be due to the fact that we were representing the support transition boundaries as the *intersection* of the supported region of configuration space with the kinematic constraints of the CS surface (see Section 2.5). Although prudent in terms of implementation complexity, the resulting coupling and lack of information regarding support of configurations not *on* the CS surface were a nuisance. Ideally, we would have liked to be able to manipulate *both* the CS surface and the surface of the supported region, as illustrated in Figure 2.12, in the full (x, y, θ) configuration space *independently* of one another.

- The requirement that modifications via apparent inversion to the CS facets and support transition boundaries could only be made in an (x, y) plane in configuration space was restrictive. Again, prudence in the implementation placed additional constraints on the flexibility of the tools. Ideally, since we visualize the motion constraints in the full (x, y, θ) configuration space, we would also like to be able to manipulate these constraints fully in the same space.

- With our motion constraint representations we have moved beyond modeling object geometry to make explicit and accessible the motion constraints implicit in the interaction of objects. Recalling Figure 4.14 we note that the ultimate objective of the design exercise is to produce desired object motions. In a sense, the motion constraints are secondary to the paths themselves. Ideally, we would like to be able to manipulate the motion paths more directly via apparent inversion to achieve a design, instead of doing so indirectly by manipulating constraints on the motions.

Finally, some general observations about the design of motion constraints gained from the above examples.

- Global coupling between design parameters and motion constraints is what makes geometric design difficult. Unlike application domains that may be decoupled into lumped parameter elements, such as electrical circuits or hydraulic networks, geometric interactions between objects are inherently coupled. Any system that attempts to model such systems accurately must capture this coupling.

- The effects of coupling were particularly apparent in the bowl feeder examples. Since vibratory feeders are sensorless, they can neither predict nor observe the orientation of the next part to enter the feeder. Therefore, since part/feeder interactions cannot be made to occur selectively, the designer must take into consideration the motion constraints produced by interactions along *all* possible motion paths. *Everything that can happen will happen.*

- In general, design for compliant assembly seems to be easier because we need focus on only a very local set of constraints. This conclusion may be somewhat misleading, however, since we have ignored a significant aspect of assembly design – getting a set of assembly motion constraints that are consistent with *whatever* other external constraints may be imposed on a part's design.

4.7 Summary

In this chapter we have taken the motion constraint representations developed in Chapter 2 and extended them in a number of ways to address the issue of design. We examined some of the difficulties of generating consistent shapes from a priori specifications of function in terms of motion constraints, and in particular we illustrated how the notion of generating functional shapes by sweeping fixed shapes along specified motion paths does not guarantee that the desired motion constraints will be achieved. We introduced the space of design parameters as a domain in which design of shape and other parameters could viewed in the context of a search. We developed a set of design functions to operate on design parameters that were divided into two classes: parametric functions, which we labeled apparent inversion, that allow us to select and consistently manipulate design parameters indirectly in the context of motion constraints, and topological functions, one example of which is an imposed out-of-plane swept motion designed to cut out contours in a supporting track.

We introduced and explored the notion of dynamic constraint visualization where manipulation of constraints and parameters: *(i)* allow us to explore the neighborhood of a point in design space and identify what changes to a design are likely to achieve a desired function, and *(ii)* keep us aware of what is and is not possible to modify independently. In particular, we noted that coupling between different constraint representations and the underlying design parameters posed a serious challenge to the design of interacting shapes.

We presented an overview of an implemented design and analysis toolkit – `cspace-shell` – that provides a computational environment to support the representation and interactive manipulation of motion constraints for design. We gave an overview of a design methodology for using this toolkit in the design of vibratory bowl feeders and compliant peg-in-hole assemblies. In developing this methodology we also introduced a number of constructs designed to organize our search of a large design space and help control the level of complexity and coupling between parameters and constraint representations. Finally, we presented a series of design examples for bowl feeders and, to a more limited extent, peg-in-hole assemblies in which we utilized and evaluated the toolkit.

Implementation

Chapter 5

5.1 Goals

The primary motivation for implementing the motion constraint based shape design system was to provide both a research tool and proof of concept demonstration for visualizing and designing function from shape. The following goals guided the multitude of choices, assumptions, and optimizations that form the resulting implementation:

- **SPEED** – Compute and recompute the set of motion constraints from shape at speeds that are sufficient for near real-time interactive feedback.

- Emphasis on the modification of shape parameters for the purposes of design. This is in contrast to existing configuration space based analysis and planning systems that emphasize the computation of motions with respect to static motion constraints derived from fixed shapes.

- Provide direct access to design parameters from within the same functional representations that are used for visualization and analysis. This includes providing as natural and intuitive an interface as possible for the manipulation of **representations** ⇔ **design parameters** by means of *apparent-inversion* functions mapping interactive user inputs in configuration space to shape.

In addition to these general goals, the implementation has been tailored for representing and manipulating motion constraints that are particularly well suited for design in the domain of vibratory part feeders, although additional features have been added for modeling and design of compliant assembly tasks.

5.2 System Overview

The interactive shape design system consists of approximately 25,000 lines of C source code compiled to run on a Silicon Graphics Personal Iris workstation. The code is organized into the following major modules:

`cspace-shell` Core routines including the interactive graphical user interface and main loop running the configuration space display for representing and manipulating motion constraints and shape.

`cs_routines` Routines to generate CS facets.

`cs_motions` Routines for the incremental computation of motion paths and local CS topology.

`cs_support` Routines to compute the approximate boundaries of supported and unsupported regions on CS facets, and generating track cutouts for imposed part motions.

`cs_selection` Routines that allow the user to select and modify part, bowl and track vertices via changes to the CS or LOS boundaries.

The interactions among the major modules is illustrated in Figure 5.1. A number of additional files provide data structure definitions, I/O support and various utilities.[1] The remainder of this chapter presents in detail the implementation of the major components of `cspace_shell` and its subroutines.

The main event loop in `cspace-shell` handles all of the functions for displaying objects and their motion constraints in configuration space. User selection of or incremental modifications to any of the shape parameters automatically cause the CS to be continuously recomputed and displayed during the modification. If enabled, the support regions on each of the facets in the CS are also recomputed and displayed. Other functions that need not or cannot be performed continuously, such as the integration or animation of motion paths, are executed once "on demand" by the user.[2]

[1]These files include: `cs_data_structures.h`, `cs_file_io`, `cs_to_iris`, `cslice_iris`, `cs_utilities` and `cs_iris_utilities`.

[2]Numerical integration of motion paths is the most computationally expensive operation performed in `cspace-shell`, and would be unable to provide acceptably fast feedback to the user during a design operation.

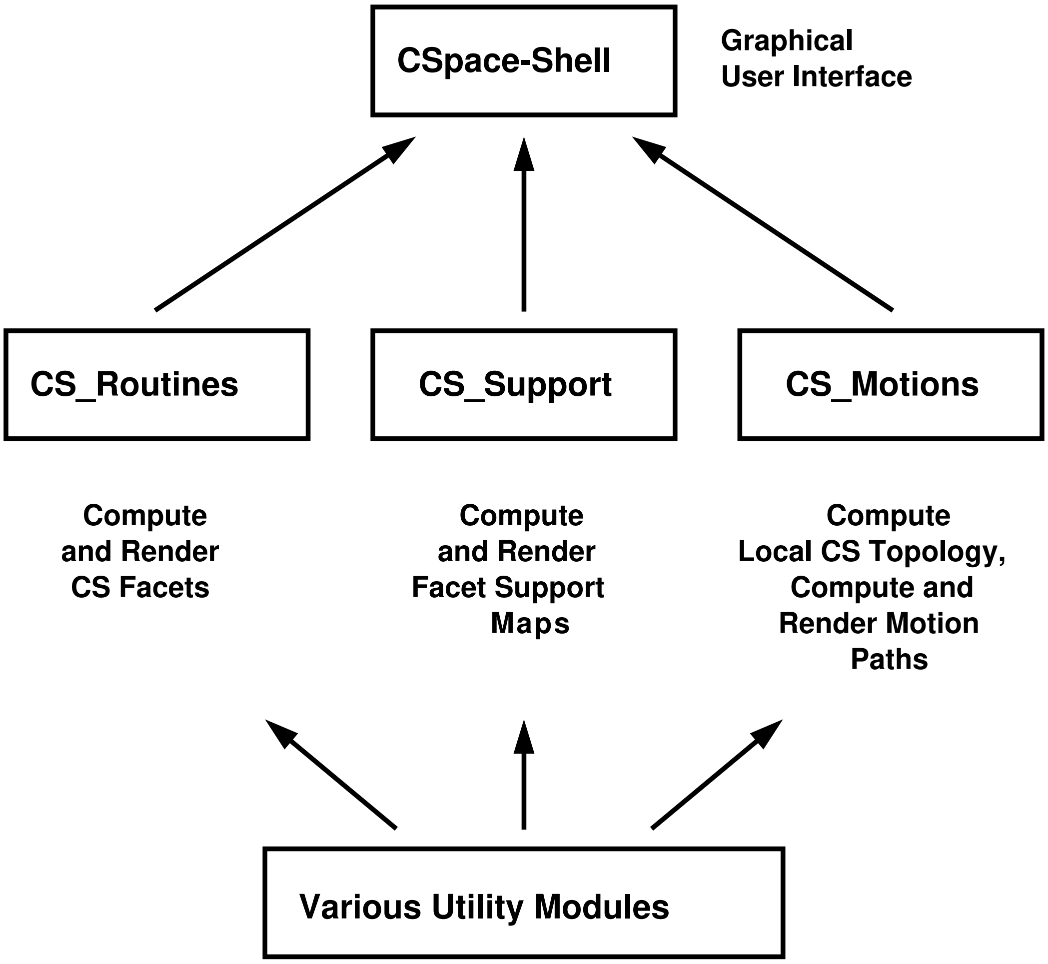


Figure 5.1: General overview of the major modules comprising the interactive motion constraint design system.

5.3 Generating Kinematic Constraints

5.3.1 Contact Facets

We explicitly model the two basic types of single-contact interactions between a pair of planar polygons as constraint *facets*:

- **type A** – an edge of Polygon A (moving) touching a vertex of Polygon B (stationary), and
- **type B** – an edge of Polygon B touching a vertex of Polygon A.

The contact generating each facet type reduces the degrees of freedom of the moving polygon from three to two. A convenient parameterization for the remaining two degrees of freedom is p and θ as shown in Figure 5.2. The p parameter determines the non-dimensional position $(0,1)$ along a polygon's edge at which a vertex of the other polygon contacts that edge. The θ parameter is the orientation of the moving polygon, and is the same θ used in the (x,y,θ) configuration space. The values of p and θ thus determine the position of a point on the surface of a facet that is itself embedded in the (x,y,θ) configuration space.

Using the parameters (p,θ) we may write the equations for a point on a facet surface. For a type A facet, we have:

$$\vec{F}^A(p,\theta) = \vec{R}_j^B - Rot(\theta)(\vec{R}_i^A + p\vec{E}_i^A) \tag{5.1}$$

$$\begin{aligned} p &\in [0,1] && (5.2)\\ \theta &\in [\theta_{min}, \theta_{max}] && (5.3) \end{aligned}$$

where $\vec{R}_j^B$ is the position vector of the jth vertex of the stationary polygon from the origin of the world coordinates, $\vec{R}_i^A$ and $\vec{E}_i^A$ are the position and edge vectors, respectively, of the ith vertex and edge of the moving polygon with respect to that polygon's reference point, and p and θ are the facet parameters. $Rot(\theta)$ is the rotational transform:

$$Rot(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Similarly, for a type B facet we have:

$$\vec{F}^B(p,\theta) = \vec{R}_j^B + p\vec{E}_j^B - Rot(\theta)(\vec{R}_i^A) \tag{5.4}$$

$$\begin{aligned} p &\in [0,1] && (5.5)\\ \theta &=\in [\theta_{min}, \theta_{max}] && (5.6) \end{aligned}$$

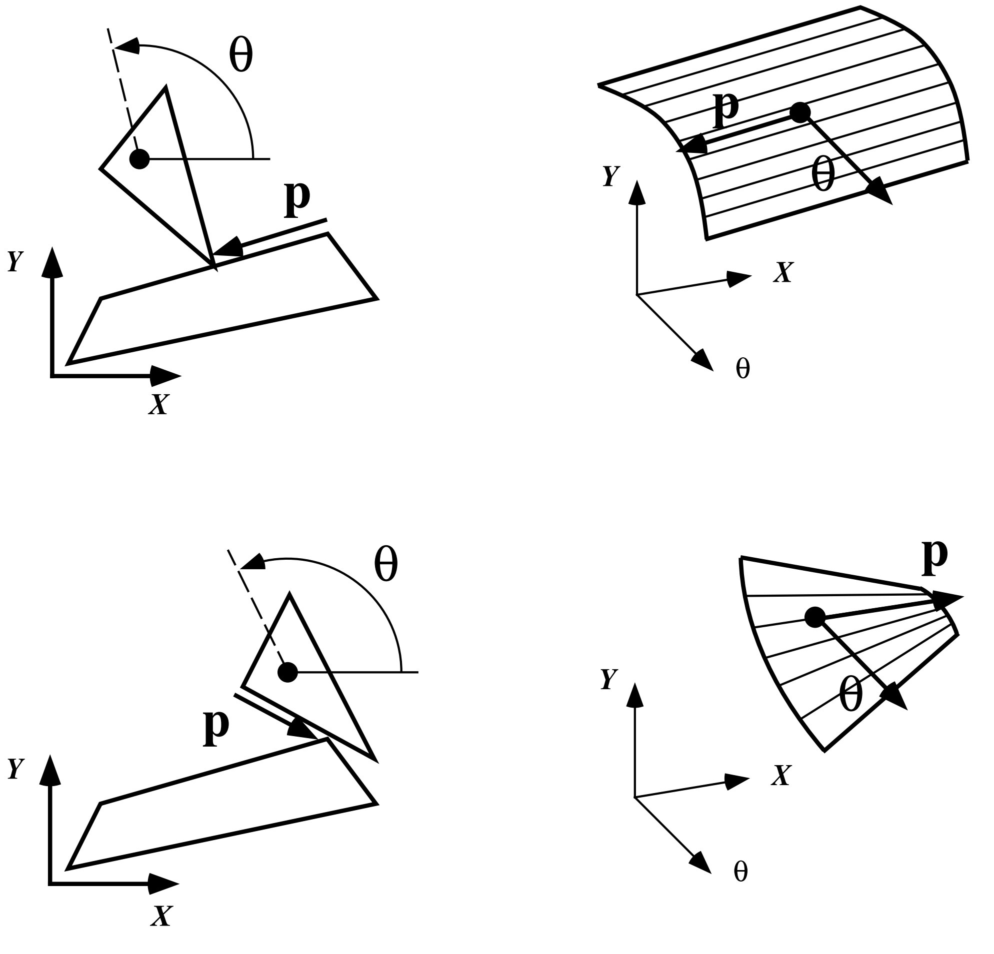
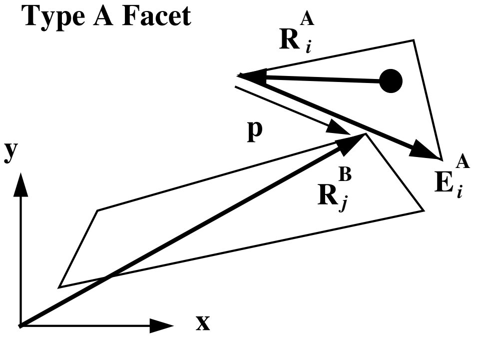


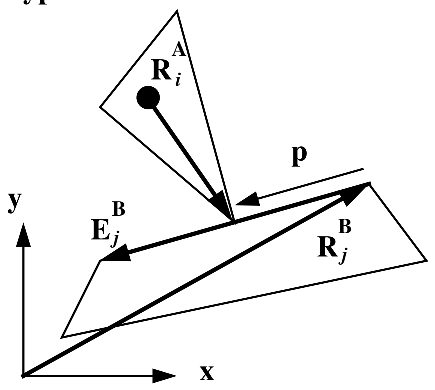
Figure 5.2: Parameterization of facets by p and θ.

Type A Facet



$$\mathbf{F}^{A}_{i,j}(\mathbf{p},\theta) = \mathbf{R}^{B}_{j} - (\mathbf{R}^{A}_{i} + \mathbf{p}\,\mathbf{E}^{A}_{i})\,\mathbf{Rot}(\theta)$$

Type B Facet



$$\mathbf{F}^{B}_{j,i}(\mathbf{p},\theta) = \mathbf{R}^{B}_{j} + \mathbf{p}\,\mathbf{E}^{B}_{j} - \mathbf{R}^{A}_{i}\,\mathbf{Rot}(\theta)$$

Figure 5.3: Vector notation for facet equations.

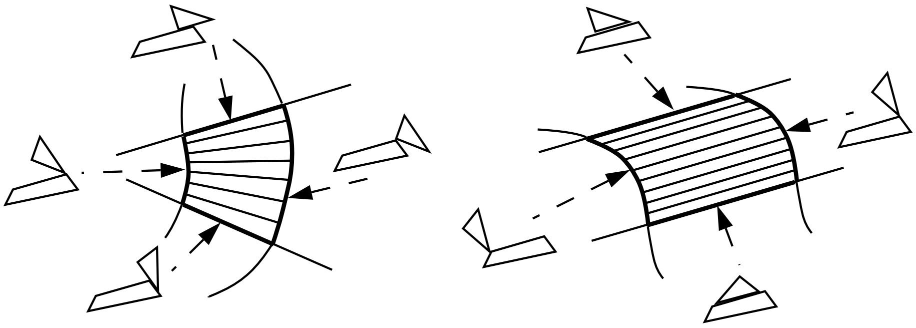


Figure 5.4: Bounding curves for type A (left) and type B (right) facets.

Figure 5.3 illustrates the vector notation for both facet types.

In terms of the configuration space parameters (x, y, θ) we may write for a type A facet:

$$x = R^B_{j_x} - ((R^A_{i_x} + pE^A_{i_x}) \cos\theta - (R^A_{i_y} + pE^A_{i_y}) \sin\theta) \tag{5.7}$$
$$y = R^B_{j_y} - ((R^A_{i_x} + pE^A_{i_x}) \sin\theta + (R^A_{i_y} + pE^A_{i_y}) \cos\theta), \tag{5.8}$$

and for a type B facet we have:

$$x = R^B_{j_x} + pE^B_{i_x} - (R^A_{i_x} \cos\theta - R^A_{i_y} \sin\theta) \tag{5.9}$$
$$y = R^B_{j_y} + pE^B_{i_y} - (R^A_{i_x} \sin\theta + R^A_{i_y} \cos\theta). \tag{5.10}$$

Contact facets are represented by ruled surfaces in the (x, y, θ) configuration space. Each facet is bounded by four curves corresponding to maximum and minimum values of the parameters p and θ. The curves corresponding to maximum and minimum p values are sinusoidal space curves in the (x, y, θ) configuration space, whereas the maximum and minimum θ curves are (x, y) line segments in fixed θ slices of configuration space. Figure 5.4 illustrates these curves for both type A and type B facets.

The orthogonal set of normal and tangential vectors at any point on a facet surface may be written in terms of (p, θ) as:

$$\vec{u} = \frac{\frac{\partial \vec{F}}{\partial p}}{\left|\frac{\partial \vec{F}}{\partial p}\right|} \tag{5.11}$$
$$\vec{v} = \frac{\frac{\partial \vec{F}}{\partial \theta}}{\left|\frac{\partial \vec{F}}{\partial \theta}\right|} \tag{5.12}$$
$$\vec{n} = |\vec{u} \times \vec{v}|. \tag{5.13}$$

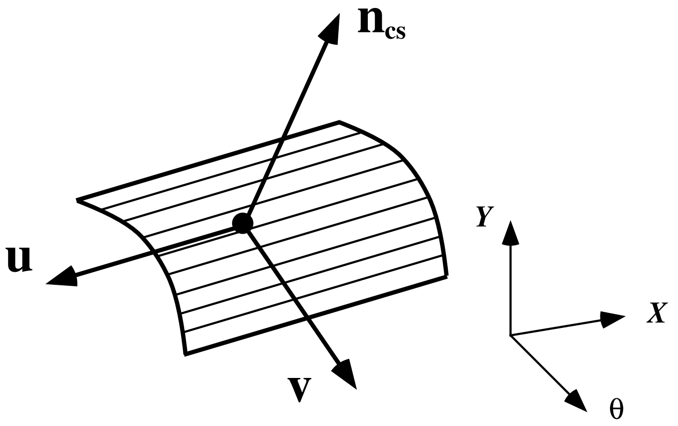


Figure 5.5: Normal and tangential vectors at a point on the surface of a facet.

For a type A facet, the expressions for the components of $\vec{u}$ and $\vec{v}$ are:

$$
\begin{aligned}
u_x &= E^A_{i_x} \cos\theta - E^A_{i_y} \sin\theta && (5.14)\\
u_y &= E^A_{i_x} \sin\theta + E^A_{i_y} \cos\theta && (5.15)\\
u_\theta &= 0 && (5.16)\\
v_x &= -(R^A_{i_x} + pE^A_{i_x}) \sin\theta - (R^A_{i_y} - pE^A_{i_y}) \cos\theta && (5.17)\\
v_y &= (R^A_{i_x} + pE^A_{i_x}) \cos\theta - (R^A_{i_y} + pE^A_{i_y}) \sin\theta && (5.18)\\
v_\theta &= 1, && (5.19)
\end{aligned}
$$

and for a type B facet:

$$
\begin{aligned}
u_x &= E^B_{i_x} && (5.20)\\
u_x &= E^B_{i_y} && (5.21)\\
u_\theta &= 0 && (5.22)\\
v_x &= -R^A_{i_x} \sin\theta - R^A_{i_y} \cos\theta && (5.23)\\
v_x &= R^A_{i_x} \cos\theta - R^A_{i_y} \sin\theta && (5.24)\\
v_\theta &= 1. && (5.25)
\end{aligned}
$$

Following the convention that p increases while traversing an edge of the polygon in a counterclockwise sense, the normal $\vec{n}$ points *outward* from the surface of the facet, as shown in Figure 5.5.

5.3.2 Traces, Moves and Turns

The complete set of contacts possible between two polygons may be obtained by enumerating all vertex and edge combinations between the two polygons, resulting in $2nm$ contacts between polygons with n and m vertices/edges respectively. A convenient representation for generating and indexing contact facets by using the interacting polygon features that created them is the *trace* due to Guibas et. al. [37]. In the trace representation, a polygonal contour is represented as the path, or trace, swept out by a penpoint that is alternately allowed to move in a directed straight line segment or to change orientation (i.e. turn) at a point. In terms of a polygon, each *move* in a trace corresponds to an edge of the polygon and each *turn* corresponds to a vertex joining two edges. A closed trace produces a closed polygon where the penpoint is returned to the same position and orientation from which it started.

By convention a trace traverses a polygon in a counterclockwise direction so that the interior of the polygon is always to the left of the trace. A convex vertex on the polygon is represented as a left turn, and a concave vertex as a right turn. A move is referred to as a *forward* move if it is traversed from start to end in the same direction as its orientation. All moves used to represent edges of polygons are forward moves.

One advantage of using the trace representation is that generating the complete set of motion constraints for two polygons (with fixed orientations) is equivalent to convolving the two polygon traces to form a new contour, which is itself a trace. Each turn in a trace sweeps out a range of angles between the move entering and the move leaving the turn. Convolving two traces involves adding the offset from the reference point to the turn of one trace to each of the moves of the other trace whose orientations lie within the angle range of the turn, and then repeating the process for each turn on the other trace. Figure 5.6 illustrates this process for two traces, along with the resulting convolved trace. Convolving two closed traces produces a trace that is itself closed.[3]

Another advantage of the trace representation is that convolutions involving convex and concave vertices are treated uniformly. In particular, convolving a forward move (polygon edge) from one trace with the left turn (convex polygon vertex) of another trace produces a forward move in the output trace, whereas convolving a forward move with a right turn (concave polygon vertex) produces a *backward* move in the output trace. A backward move is traversed from its endpoint to its start-point rather than vise versa. An important property of backward moves is that they only appear in the interior of the closed output trace formed by the convolution of two closed input traces. In other words, backward moves correspond to motion constraints that are unreachable from the exterior of a closed set of constraints. This is a useful feature because once we determine that a constraint edge is backward we do not have to compute any more detailed information about it since it cannot be

[3]See Guibas et. al. [37] for more details about 2D traces and their properties.

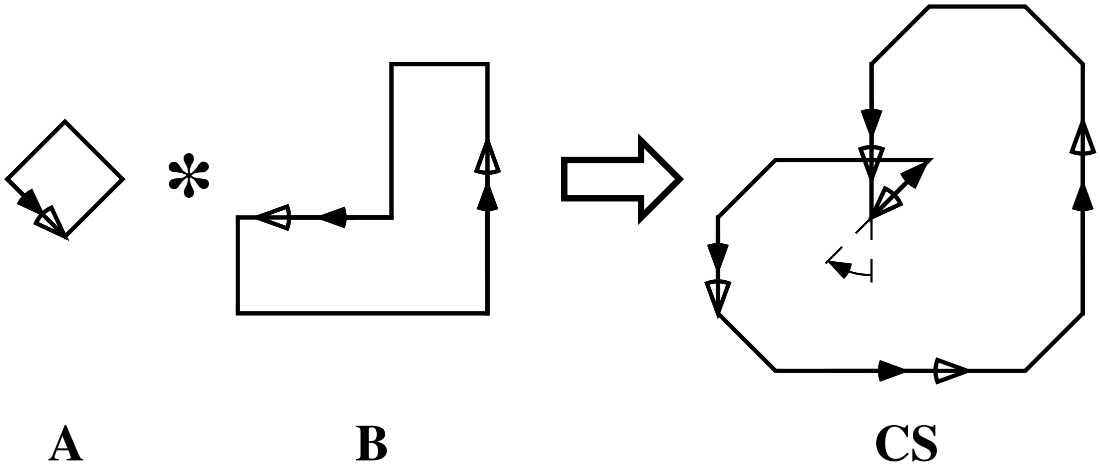


Figure 5.6: Planar polygons represented as traces, and the trace corresponding to their convolution.

on the surface of the CS.[4]

When one of the planar polygons is allowed to rotate, as is the case in `cspace-shell`, we use a generalization of the trace representation in which each of the facets described earlier corresponds to a move of the convolved trace that is swept in the θ dimension of the (x, y, θ) configuration space over the entire angle range for which that move is valid. Specifically, a facet is a move (edge) of one polygon that is offset by the position of a turn (vertex) of the other polygon over the entire range of angles for which the move's orientation lies within the turn's angle range. Using the trace notation, a facet is identified by a type (type A or type B), an index to a trace move, and an index to a trace turn. This type-move-turn indexing scheme is particularly useful in determining the topological relationships between facets forming the CS in configuration space. Equations for a facet expressed as turns and moves can be derived from equations 5.1 and 5.4 by substituting a **turn** for the $\vec{R}_i^{A/B}$ and a **move** for $\vec{E}_i^{A/B}$.[5]

Enumerating and computing the complete set of individual constraint facets for two interacting polygons involves a relatively small amount of computation; the overall complexity, both in size and time, is $O(n^2)$ where n is the typical number of edges/vertices in each of the polygons. Computing the intersections and topological

[4]We will still enumerate facets corresponding to backward moves in our CS data structure because they provide topological closure of the set of facet in configuration space and are useful in determining facet adjacency.

[5]In `cspace-shell`, the moving polygon is reflected through the origin before being converted to a trace, so the "−" sign in equations 5.1 and 5.4 would be replaced with a "+" sign.

relationships between all facets in the CS, however, is generally a more difficult and computationally expensive proposition.[6] Fortunately, for the purpose of representing the CS graphically it is sufficient to compute only the complete set of individual facets. For the purpose of computing object motions, however, we will need to compute at least those topological features on the CS that constrain specific motions. In the next section we will describe these computations in more detail along with the procedures for computing specific object motions.

In summary, the CS is constructed as an array of individual facets representing contacts between edge and vertex feature pairs of two polygons. There is no explicit representation of the relationships between the facets in the CS for the purposes of rendering. This is done in order to enhance the speed at which CS surfaces may be generated, rendered and manipulated. Specifically, by utilizing the depth buffer[7] of the Iris workstation to hide those portions of the facets that are occluded by other facets, we are be able to generate and render a detailed graphical image of the CS in near real-time, as is necessary for implementing the kind of interactive manipulation features discussed in earlier chapters.

5.4 Computing Motions

Part motions are represented as paths in configuration space, approximated at the lowest level by a set of discrete points. Specifically, a motion path is represented in `cspace-shell` as an array of contact states, where each contact state contains an array of those points along a path that are in contact with the same set of facets. Specifically, a contact state encapsulates that portion of a path that is in contact with the same set of facets corresponding to contacts between topologically distinct pairs of features of the moving and stationary objects. A contact state may contain a point or set of points in contact with a single facet, an edge between two facets, a vertex formed by three facets, or no facets at all (i.e. a free or unconstrained motion).[8]

The task of computing paths in `cspace-shell` is divided into two distinct processes: determining the local topology of the CS and piecewise numerical integration

[6]The worst case complexity is $O(n^4)$.

[7]The depth buffer, or Z-buffer, is a relatively standard piece of graphics hardware in which the distance from the viewing plane to each point of a rendered object is computed and used to determine whether or that point should be drawn. A point on an object being drawn, such as a contact facet, that is further away from the viewing plane than a previously drawn point with the same (x, y) screen position, will not be drawn. In this way, overlapping or occluded objects are drawn correctly and quickly.

[8]In those cases where, due to a geometric coincidence, more than three facets can come into contact at a point, the extraneous are removed from the contact state. See the description of `detect_new_contacts()` in Section 5.4.1.3.

of individual motion paths.

5.4.1 Computing Constraint Set Topology

The basic element for representing constraints in `cspace-shell` is the individual contact facet. Features on the CS surface formed by interactions between multiple facets, such as edges or vertices, are represented *implicitly* as relationships between two or more individual facets within each contact state during path integration. One of the contact facets in each contact state is labeled as the reference facet for that state. This reference facet, or *ref_facet* is used to store an array of indices to the other facets in the CS that a motion path may encounter when integrating from the current position.[9]

There are two categories of facet that may be encountered during the integration of a motion path: an adjacent facet and an intersecting facet.

5.4.1.1 Facet Adjacencies

An adjacent facet, as the name implies, is a facet that is adjacent to one of the four boundaries of the ref_facet as determined by the limiting values of the facet parameters $p = (0, 1)$ or $\theta = (\theta_{min}, \theta_{max})$. Facet adjacencies correspond to local contact transitions of adjacent features of the polygons in contact.

Each of the four boundaries of a ref_facet is adjacent to at least one and at most two other facets. The first, so-called primary adjacent facet, is the contact facet corresponding to the transition of contacts between consecutive polygon features. The primary adjacent facet is always of the opposite type from the ref_facet, and is always adjacent to the ref_facet. The secondary adjacent is a another facet that is a adjacent to the primary adjacent facet, and may also be adjacent to the ref_facet if certain conditions hold. The secondary adjacent facet is always of the same type as the ref_facet.[10]

For an adjacency along a ref_facet's $p = 0$ or $p = 1$ boundary, the secondary adjacent will be adjacent to the ref_facet iff the angle range $[\theta_{min}, \theta_{max}]$ of the ref_facet overlaps that of the secondary adjacent facet. Similarly, the ref_facet's $\theta = \theta_{min}$ or θ_{max} boundary will be adjacent to both the primary and secondary adjacent facets iff the x, y length of the ref_facet, i.e. the length of the ref_facet's corresponding polygon edge, is shorter than the length of the primary adjacent facet. Figure 5.7 illustrates some typical adjacency cases and the polygon features that generate them. Table 5.1

[9] An exception is the free contact state, which has no facets in contact. A temporary null-facet data structure is generated and used to store the information necessary to handle such cases.

[10] Actually, the primary adjacent facet represents a contact between the same polygon vertex (turn) and either the previous or subsequent polygon edge (move), whereas the secondary adjacent represents a contact between the same polygon edge (move) and either the previous or subsequent polygon vertex (turn).

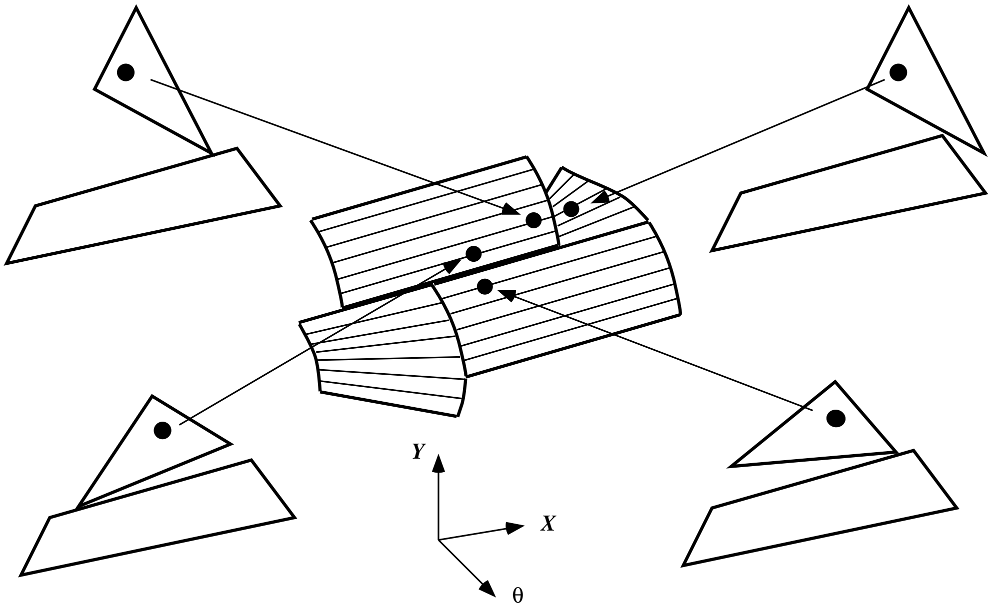


Figure 5.7: Example facet adjacencies and their corresponding polygon feature transitions.

enumerates the possible adjacencies for both a type A and type B ref_facet in terms of that facet's type, move index and turn index.

Topologically, a $p = 0$ or $p = 1$ facet adjacency represents a vertex-vertex contact between the moving and stationary polygons, and forms a space curve in the (x, y, θ) configuration space. A θ_{min} or θ_{max} adjacency between two facets corresponds to an edge-edge contact between the two polygons, and forms a straight line parallel to the (x, y) plane in the configuration space. In both types of adjacency, the moving polygon has only one degree of freedom.

5.4.1.2 Facet Intersections

Some contact facets that are locally feasible may correspond to contacts that are globally unreachable due to obstructions by other polygon features. The CS for polygons with concavities, like that for purely convex polygons, is made up of facets that are locally adjacent. However, because some of the facets are *backwards* facets formed by edges convolved with concave vertices, the CS "wraps in" on itself, resulting in facets that are partially or fully imbedded within the CS and are therefore unreachable because they are occluded by other facets. In such cases, some facets intersect other facets on the CS, as illustrated in Figure 5.8.

Because facet adjacencies are due to transitions between consecutive polygon

Adjacencies for facet: (A, m, t)		
Adjacency Type	*1st Adjacent*	*2nd Adjacent*
$p = 0$	(B, t, m^-)	(A, m^-, t)
$p = 1$	(B, t^+, m)	(A, m^+, t)
θ_{min}	(B, t, m)	(A, m, t^-)
θ_{max}	(B, t^+, m^-)	(A, m, t^+)
Adjacencies for facet: (B, m, t)		
Adjacency Type	*1st Adjacent*	*2nd Adjacent*
$p = 0$	(A, t, m^-)	(B, m^-, t)
$p = 1$	(A, t^+, m)	(B, m^+, t)
θ_{min}	(A, t^+, m^-)	(B, m, t^+)
θ_{max}	(A, t, m)	(B, m, t^-)

Table 5.1: Table of adjacencies for type A and type B facets indexed by *(type, move-index, turn-index)*. For example, the 1st and 2nd $p = 0$ adjacent facets to facet $(A, 1, 2)$ would be facet $(B, 2, 0)$ and facet $(A, 0, 2)$, respectively.

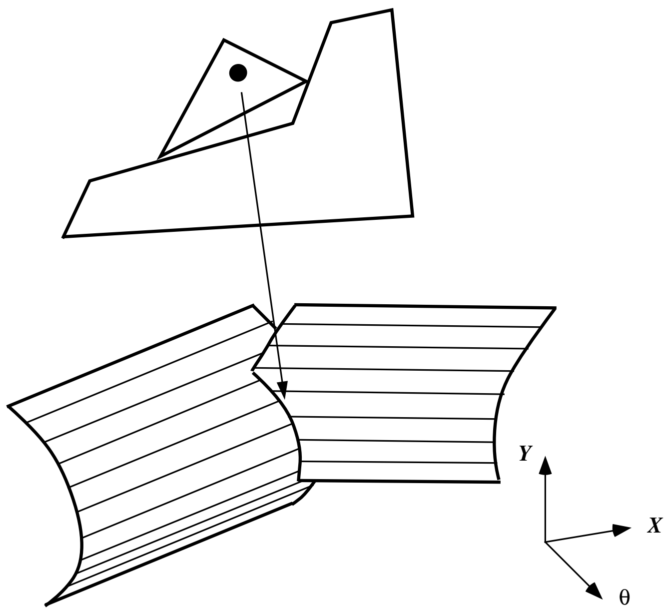


Figure 5.8: Intersecting facets and their corresponding polygon feature contacts.

features, the entries in Table 5.1 are sufficient to determine the complete set of adjacency relationships for any facet on the CS. For intersections between facets, however, we must explicitly test each facet pair to determine if an intersection is possible. Specifically, when creating a contact state we must test the ref_facet against every other facet in the CS and record the indices to those facets with which an intersection may occur. To make this process faster and more efficient, we employ a series simple tests to determine whether intersection between a given facet and the ref_facet is possible. These tests are summarized as follows:

1. **Are the facets adjacent?** Adjacent facets cannot intersect one another.
2. **Do the $[\theta_{min}, \theta_{max}]$ ranges overlap?** Facets with disjoint θ ranges cannot intersect.
3. **Do the (x, y) bounding boxes of those portions of the facets within the same θ range intersect?** Facets with non-intersecting bounding boxes cannot intersect. We compute the (x, y) bounding boxes by determining the minimum and maximum (x, y) values of the $p = 0$ and $p = 1$ space curves bounding each of the facets within their common θ range.

Facets that pass these tests can, but do not necessarily, intersect one another and are listed in an array labeled *intersects* in the ref_facet data structure.[11]

5.4.1.3 Monitoring Contact Transitions

Once we have the two arrays of adjacent and intersecting facets computed and stored within the ref_facet of a contact state, we can determine if any new contacts are encountered during incremental motion integration simply by checking each new position against those facets. We determine if a new contact has been made with an adjacent facet simply by monitoring the values of the facet parameters p and θ to see if they are within the appropriate ranges for the ref_facet. If, for example, after integrating an incremental motion the value of the ref_facet's p parameter were to change from 0.96 to 1.03, then the motion would have encountered the facet adjacent to the $p = 1$ boundary of the ref_facet.

To determine if contact has been made with an intersecting facet during an incremental motion, we check to see if the current and previous path positions are on opposite sides of any of the facets in the intersects array. To record the relationship of a position to an intersecting facet, we introduce a new data structure called a

[11] In the present implementation, we do not bother to determine if a *can intersect* facet actually *does intersect* the ref_facet. Although it is relatively straightforward to determine if such an intersection exists by numerically solving for the roots of the difference of the two facet equations, we do not bother doing so as the associated computation would typically exceed the overhead of simply assuming that the facets do intersect.

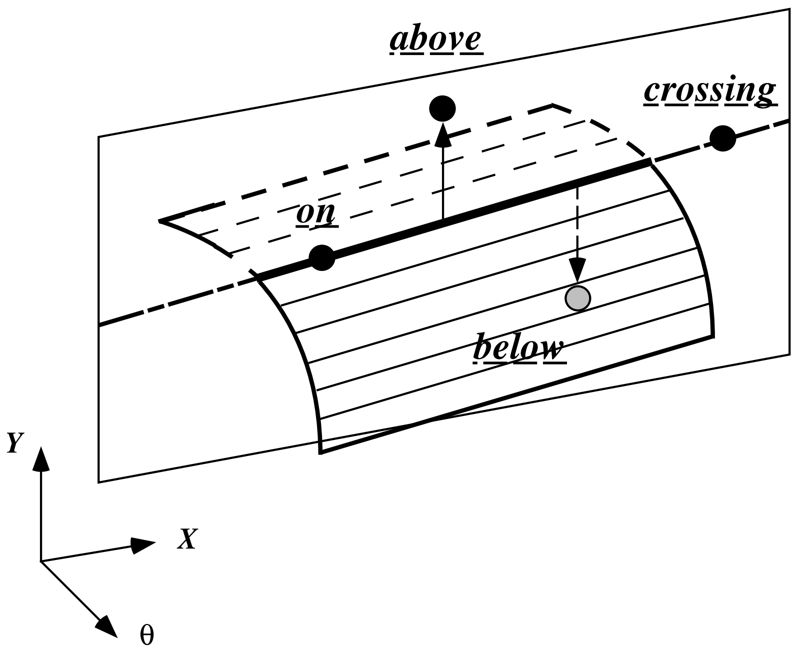


Figure 5.9: Four proximal cases for a type B intersect facet. All four points shown have the same θ component.

proximal. A proximal for an intersecting facet records the signed (x, y) distance of an (x, y, θ) point in configuration space to a line segment corresponding to a slice of the facet surface at that value of θ, as well as a flag indicating whether the point is: *above* (i.e. outside) the facet surface, *crossing* the line containing the facet slice (but outside of the slice segment's endpoints), *on* the line containing the facet slice (within the segment's endpoints), or *below* (i.e. inside) the facet surface.[12] A transition of a proximal flag from *above* to *below* or from *above* to *on* would indicate that the incremental motion has crossed that proximal's corresponding intersecting facet. Figure 5.9 illustrates the four states of a proximal flag for an intersect facet.

When a new contact is made, a new contact state containing the larger set of contact facets must be generated. Specifically, the following steps are taken if, after an integration step, changes in one or more facet parameter values or proximal flags indicate that a new contact has been made:

1. Interpolate between the startpoint and endpoint of the integration step using the previous and current facet parameter or proximal distance values to determine the (x, y, θ) point(s) at which facet(s) were crossed.

[12]The flags *crossing* and *on* are assigned to points that are within a set minimum distance (positive or negative) of the line containing the facet slice.

2. For multiple facet crossings, choose the one point that is closest to the starting point of the integration step.[13]

3. Generate a new contact state containing an array of contact facets including both the previous state's contact facets and the newly crossed facet.

4. Record the interpolated position as the *last* position in the previous contact state and the *first* position in the new contact state.

New contact states are generated during path integration either when a new contact is made, as described above, or when an existing contact is broken. The latter case is described in the next section on numerical path integration. The topology of the CS is computed locally for a ref_facet in the form of the intersects and adjacents arrays on an "as needed" basis during motion integration. Once computed, this information remains in the ref_facet's data structure until a shape modification necessitates the recomputation of the entire CS data structure. Therefore, subsequent path integrations involving the same ref_facet, such as recomputing a path after modifying one or more dynamics parameters, will generally require less computation and execute faster.

Unlike the arrays of adjacents and intersects stored in the ref_facet data structure, the array of proximals are generated and maintained only temporarily within the currently active contact state. All information regarding a point's global relationship to the surface of the CS is determined solely with respect to the facets listed within the adjacents and intersects arrays of the current ref_facet. Since no global information about the point's relationship to the overall surface of the CS is available, all paths must start either *on* or *outside* the overall CS surface in order to be integrated properly. A path that is integrated starting from a point inside the CS will cause the integration of that path to terminate with an error.

5.4.2 Numerical Path Integration

Once the local topology has been determined for a contact state by filling in the ref_facet's intersects and adjacents arrays, we may compute the set of points along the motion path using numerical integration.

5.4.2.1 Mechanics of Motion

The dynamics parameters used to compute object motions are the externally applied (x, y) force vector $\vec{F}_{ext}$, the static coefficient of friction μ, and the radius of gyration

[13]In some cases there may be a number of facets that coincide, in which case more elaborate steps must be taken to choose the proper one facet. These cases, which [13] refers to as *mc0-edges*, result from coincidental alignments between geometric features on the two polygons.

ρ of the moving object. With these parameters we can compute the reaction forces and static force equilibrium conditions for any point in a contact state. To combine the units of force and torque consistently, we must first scale the torque components of the reaction forces with ρ as:

$$F_z = \frac{\tau_\theta}{\rho}.$$

Similarly, we scale the θ components of the facet vectors with:[14]

$$\hat{u}_z = \rho u_\theta.$$

where $\hat{u}_\theta$ was introduced in Section 5.3.1.

Computing an incremental motion step involves the following basic steps:

1. Compute the configuration space friction cone at the current position in the contact state representing the set of possible reaction forces.

2. Use the friction cone to compute the reaction force for the given applied force.

3. Compute the instantaneous direction of motion using the net resultant force (if any).

4. Determine the appropriate integration step size and compute the incremental motion.

The detailed implementation of these four steps for computing motions in a *free*, *facet*, *edge*, and *vertex* contact state follows.

Friction Cone Construction

To determine the set of possible reaction forces we construct the (x, y, z) configuration space friction cone for each contacting facet. The friction cone for a single facet is shown in Figure 5.10, where $\vec{n}$ is the surface normal of the facet at the contact point, and $\vec{F}_{r1}, \vec{F}_{r2}$ are the two extremal reaction forces that together span the range of reaction possible forces. The equations for $\vec{F}_{r1}, \vec{F}_{r2}$ are:

$$\begin{aligned}
F_{r1_x} &= \alpha(n_x \cos\phi + n_y \sin\phi) && (5.26)\\
F_{r1_y} &= \alpha(-n_x \sin\phi + n_y \cos\phi) && (5.27)\\
F_{r1_z} &= \alpha(n_z \cos\phi + \beta \sin\phi) && (5.28)\\
F_{r2_x} &= \alpha(n_x \cos\phi - n_y \sin\phi) && (5.29)\\
F_{r2_y} &= \alpha(n_x \sin\phi + n_y \cos\phi) && (5.30)\\
F_{r2_z} &= \alpha(n_z \cos\phi - \beta \sin\phi) && (5.31)
\end{aligned}$$

[14] To ensure that the facet vectors remain unit vectors, we normalize them *after* introducing the scaling factor ρ.

where

$$\sin\phi = \frac{\mu}{\sqrt{1+\mu^2}} \tag{5.32}$$

$$\cos\phi = \frac{1}{\sqrt{1+\mu^2}} \tag{5.33}$$

$$\alpha = \frac{1}{\sqrt{(n_x^2+n_y^2)}} \tag{5.34}$$

$$\beta = \sqrt{\left(\frac{r_c}{\rho}\right)^2(1-n_z^2)-n_z^2} \tag{5.35}$$

and where μ is the coefficient of friction, (n_x, n_y, n_z) are the normalized components of the scaled facet normal, ρ is the radius of gyration, and r_c is the distance from the **cg** of the moving polygon to the point of contact. For a type B facet r_c is a constant measured from the **cg** to the vertex of the moving polygon that is in contact with edge of the stationary polygon, and is given by:

$$r_c = \left|\vec{R}_j^A\right| \tag{5.36}$$

where j is the *turn* index of the facet. For a type A facet the value of r_c is a function of the facet parameter p and is given by:

$$r_c = \left|\vec{R}_{i-1}^A + p\vec{E}_i^A\right| \tag{5.37}$$

where i is the *move* index of the facet.

For contact states with multiple contact facets it will be necessary to combine the friction cones for the individual facets to form the compound friction cone. For an edge contact state (two facets) we form the compound friction cone from the convex hull of the extremal friction cone reaction forces $\vec{F}_{r1}$ and $\vec{F}_{r2}$ for the two facets. Specifically, we order the four extremal force vectors to form the edges of a convex polyhedral cone as shown in Figure 5.11. We do not form the compound friction cone for a vertex contact because, by definition, a vertex has zero degrees of freedom and hence no sliding motion is possible. We do, however, check for possible motions along the three edges forming the vertex to see if a motion out of the vertex is possible, as described in Section 5.4.2.2. Finally, for a free contact state (no facets) there can be no reaction forces and hence no friction cone.

Computation of Reaction Forces

After constructing the friction cone for a contact state, we use it to compute the reaction force due to the applied force. The reaction force for a single facet contact is determined by projecting the negated applied (x, y) force onto the $(x, y, \{z = \rho\theta\})$ plane containing the configuration space friction cone. If the projected force vector is between the two extremal friction cone reaction force vectors then the reaction

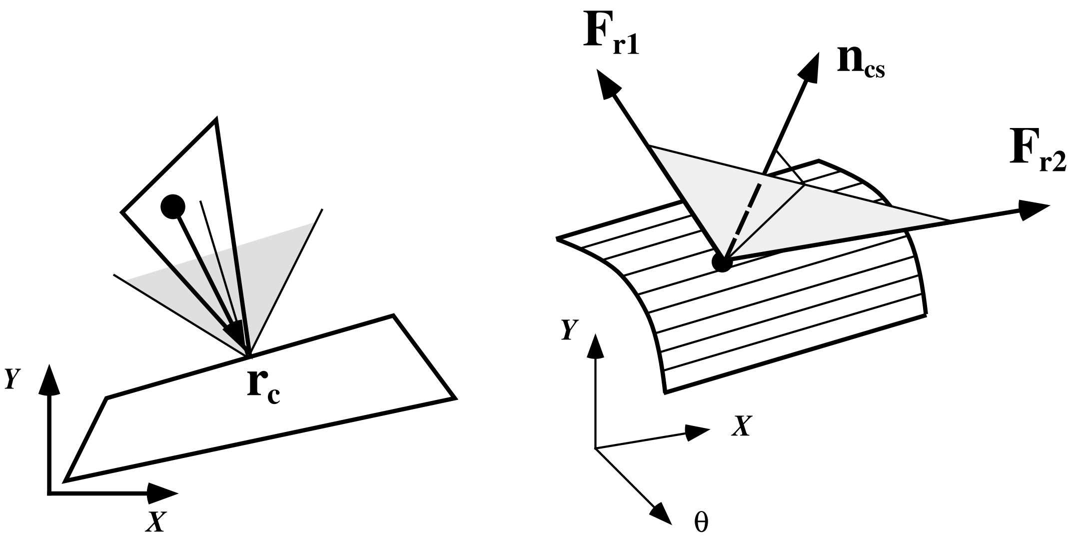


Figure 5.10: The $(x, y, \{z = \rho\theta\})$ configuration space friction cone for a single facet contact.

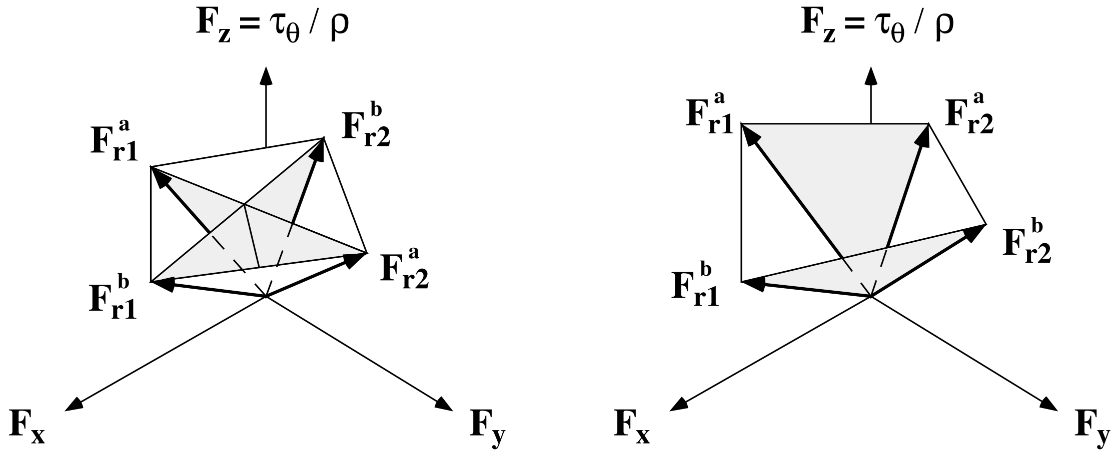


Figure 5.11: The compound configuration space friction cone for an edge contact between two facets.

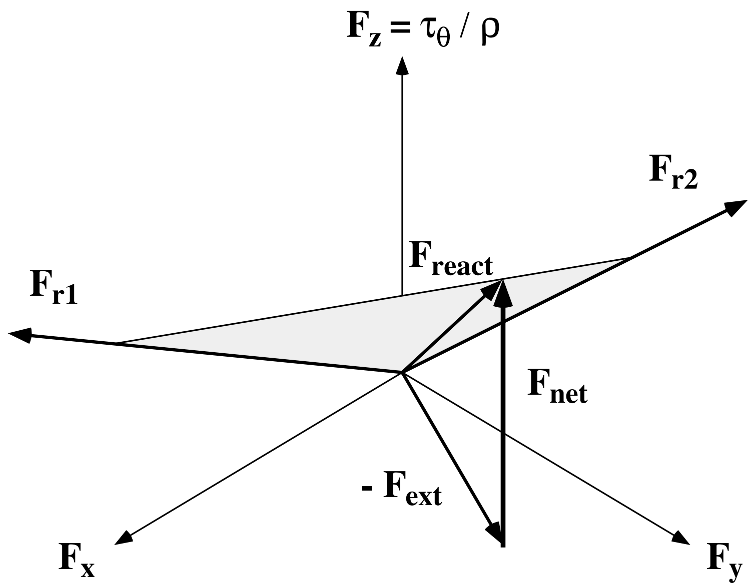


Figure 5.12: Computing the reaction force with the single facet friction cone.

force is the projected force vector. If the projected force vector is not between the two extremal force vectors, then the reaction force is equivalent to the projection of the projected force onto the closer of the two extremal force vectors. This process is illustrated in Figure 5.12.

To determine the reaction force for an edge contact, we first check to see if the applied force is pointing into the compound cone by taking the dot-product of the force with the outward pointing normals of each of the four bounding planes of the convex cone. If all four dot-products are positive, the applied force points into the cone interior and the reaction force is exactly the negated applied force. If one or more of the dot-products is negative, then the applied force lies outside of the compound polyhedral cone, an we must project the negative applied force onto the surface of the cone. Specifically, we project the negated applied force onto each the surfaces of the cone with which the dot-product was negative, and proceed in a manner similar to that for the single facet friction cone.

Computation of Instantaneous Motion Direction

To compute the force equilibrium state at the contact point we compute the net force by taking the difference between the applied and reaction forces. A non-zero net force vector determines the instantaneous direction of motion from the current contact point. For a free contact state there is no reaction force so the net force is simply the applied force. For a single facet contact state a non-zero net force will be tangent to the facet surface, and the non-zero net force for an edge contact will be parallel to the edge vector formed by the cross-product of the two facet normals at the contact point.

Computation of Integration Step Size and Incremental Motion

Following our assumption that quasi-static effects dominate the dynamics of object motion, we may write the following equations of motion:

$$\begin{aligned} F_x &= m\ddot{x} \approx \lim_{\Delta t \to 0} m \frac{\Delta x}{\Delta t^2} && (5.38) \\ F_y &= m\ddot{y} \approx \lim_{\Delta t \to 0} m \frac{\Delta y}{\Delta t^2} && (5.39) \\ F_z = \frac{\tau_\theta}{\rho} &= \frac{I}{\rho}\ddot{\theta} \approx \lim_{\Delta t \to 0} \frac{I}{\rho} \frac{\Delta \theta}{\Delta t^2} && (5.40) \end{aligned}$$

where (F_x, F_y, F_z) are the components of the net force at the point of contact, and Δt is a time step that will chosen to scale the magnitude of the incremental motion appropriately.[15]

A motion in a free contact state will follow a straight line parallel to the applied force in the absence of any externally applied torques, which we have assumed in our model. An incremental motion in a single facet contact state will be in the direction of the net force, but should also remain on the surface of the facet in the course of the motion even though the surface may be curved. To ensure this, and at the same time avoid introducing linearization errors, we resolve the net force into components expressed in the $\vec{u}$ and $\vec{v}$ tangent vectors of the facet surface, and integrate the incremental motion in terms of the p and θ facet parameters.

To distinguish between displacements in translation and rotation, we denote the combined displacement in x and y as $\overline{xy}$, which can be expressed in terms of the facet parameter p as $\overline{xy} = lp$ where l is the length of the polygon edge forming the contact facet. Substituting $\rho m = \frac{I}{\rho}$ into the above equations we can write the rotational and translational displacements in terms of the facet parameters as:

$$\begin{aligned} \Delta p &= \frac{1}{l}\left(\frac{\Delta t^2}{m}\right) F_{\overline{xy}} && (5.41) \\ \Delta \theta &= \frac{1}{\rho}\left(\frac{\Delta t^2}{m}\right) F_z. && (5.42) \end{aligned}$$

To compute the instantaneous direction of motion we may normalize Δp and $\Delta \theta$

[15] Due to the quasi-static assumption, the magnitude of the integration step may not be determined directly from the equations of motion, as would be the case for standard Euler or Runge-Kutta numerical integration techniques. Rather, the integration step size will be derived from a consideration of the errors produced by straight line approximations to motions along the surfaces of curved contact facets. Hence, integration step size, like the friction cone, is computed directly from contact facet geometry.

with respect to a third variable ΔT using

$$\Delta T^2 = \Delta p^2 + \Delta\theta^2 \tag{5.43}$$

$$\Delta T = \sqrt{\Delta p^2 + \Delta\theta^2}. \tag{5.44}$$

The normalized displacements then become

$$\frac{\Delta p}{\Delta T} = \frac{\Delta p}{\sqrt{\Delta p^2 + \Delta\theta^2}} \tag{5.45}$$

$$\frac{\Delta\theta}{\Delta T} = \frac{\Delta\theta}{\sqrt{\Delta p^2 + \Delta\theta^2}}. \tag{5.46}$$

Equations 5.45 and 5.46 allow us to express the instantaneous direction of motion on a facet surface in terms of p and θ. Because the dynamics are quasi-static, the magnitude of the incremental motion step is somewhat arbitrary. Since we are integrating linearly in the parameter space of the facet surface, the deviation of the motion from a straight line in $(x, y, \{z = \rho\theta\})$ configuration space will depend on the degree of curvature of the surface along the direction of motion. Hence, to bound the maximum deviation from a straight line we select the integration step size based on the curvature metric given by:

$$\Delta T_{max} = \sqrt{\frac{2E_{max}}{\kappa_n} - E_{max}^2} \tag{5.47}$$

where E_{max} is the maximum allowable error in $(x, y, \{z = \rho\theta\})$, and κ_n is the curvature of the facet along the motion path, which is derived for type A and type B facets in Appendix B. Using this metric, we compute the motion step for a facet in p and θ as:

$$\Delta P_{max} = \left(\frac{\Delta p}{\sqrt{\Delta p^2 + \Delta\theta^2}}\right) \Delta T_{max} \tag{5.48}$$

$$\Delta\theta_{max} = \left(\frac{\Delta\theta}{\sqrt{\Delta p^2 + \Delta\theta^2}}\right) \Delta T_{max} \tag{5.49}$$

and compute the new (x, y, θ) position in configuration space using the facet equations.

An incremental motion in an edge contact state will be in the direction of the net force along the edge vector, which like the single facet contact state may be curved. Rather than attempting to derive a parameter to describe motion along the curve forming the edge, we will simply integrate the motion along the edge vector expressed in terms of the $\vec{u}$ and $\vec{v}$ tangent vectors of both facets. After integrating the two incremental facet motions individually, we choose the smaller of the two in terms of total distance in (x, y, θ) configuration space for the edge motion, and remove any linearization error by projecting the new point onto the edge between the facets calculated at the new value of θ.[16] This is a rather crude approximation to the actual edge motion, but it eliminates having to compute the incremental motion along an edge curve that might be difficult or impossible to compute in closed form. Since intersection curves between facets may have a much higher degree of curvature than the facets themselves, this approximation is a recognized weak point of the current motion path integration implementation.

5.4.2.2 Selecting Consistent Motions

The computations for integrating motion in each of the four types of contact state (free, facet, edge, vertex) assume that resulting motion will maintain all of the contacts in that state. For example, computing the motion in a single facet contact state in terms of the facet parameters p and θ does not take into account the possibility that the given applied force will cause the motion to leave the facet surface. To ensure that the computed incremental motions are consistent, we must check to make sure that the assumptions implicit in the equations of motion are in fact valid.

A free motion, by definition, involves no contacts and hence is always consistent given a non-zero applied force. A facet motion is consistent if the applied force points *into* the facet surface at the initial position of the integration. This may be checked by computing the dot-product of the applied force vector with the facet's normal vector at that point. A negative value indicates that the force points into the facet surface and any motion must take place on the facet, whereas a positive result indicates the motion will leave the facet surface.

An edge motion is consistent if the motions due to the applied force on the two facets forming the edge both point into that edge. We may check this by computing the vector tangent to each facet's surface that is perpendicular to the edge vector and taking the dot-product of the corresponding facet motions with each of these vectors. A positive value for either dot-product would indicate that the resulting motion would break contact with the other facet and become a single facet motion.

Finally, we can determine if a motion into or out of a contact state for a vertex formed by three facets is possible by checking if a motion along any of the three edges intersecting at the vertex points away from the vertex. If all three edge motions point into the vertex then no motion is possible and integration is terminated. Otherwise a consistent edge motion must be selected.

In each of the above cases, if the incremental motion in the current contact state is found to be inconsistent, we generate a new contact state with the next lower number of contacts (one greater degree of freedom), and store the current position as the **last**

[16] If the edge is a straight line, such as a θ_{min} or θ_{max} adjacency, then there the new position has the same θ component as the old position, and no linearization is necessary.

point in the previous state and the **first** point in the new state. Figure 5.13 illustrates the above motion selection procedures for the four types of contact state.

5.4.2.3 Computing Initial Conditions

There are two types of initial conditions used for integrating motions in `cspace-shell`, the choice of which depends on the application domain. To find the initial conditions for part motions through a vibratory feeder, we identify the set of stable initial orientations of the moving object in contact with the bowl wall. We construct this set, which corresponds to the possible orientations in which a part may enter the feeder, by first taking the convex hull of the moving polygon and generating an edge contact state for each pair of vertices on the convex hull. Each contact state contains the pair of contact facets corresponding to a contact between one of the two vertices and the top leftmost horizontal edge of the stationary polygon along which parts enter the feeder. The stability of each edge contact state is then evaluated by computing the reaction forces at the two contact points that are necessary to maintain static equilibrium with an applied force through the polygon's **cg** pointing along the $-y$ axis, as illustrated in Figure 5.14. If one of the reaction forces is negative $(-y)$, then the orientation is unstable, and the contact state is removed.

For feeder design purposes, it would be helpful to know the probability with which a part in a given orientation will enter the feeder. We compute the probability associated with each stable contact state by assuming a uniform probability distribution over the $\theta = (0 \rightarrow 2\pi)$ range of orientations in which a part may first come into contact with the bowl wall, and then dividing this range into θ regions in which the part would tend toward a given stable orientation. The widths of the resulting θ ranges are then divided by 2π to obtain the probability $(0 \rightarrow 1)$ of a part appearing in that orientation. Figure 5.15 illustrates this process using the radius function of a part.

To integrate more general object motions, such as a part being placed into a subassembly, we allow the user to select an arbitrary (x, y, θ) position corresponding to a *free* contact state, or a point on a facet surface corresponding to a *facet* contact state. The free (x, y, θ) position is selected using three position sliders, and is graphically displayed in configuration space as the intersection of three lines parallel to the (x, y, θ) axes. The free contact state generated for this position uses a temporary null facet data structure containing all of the facets in the CS within the *intersects* array.[17] The user is responsible for selecting a position that is on the **outside** of the CS.

A position in a facet contact state is selected by having the user click the mouse over a displayed facet. The position is mapped from (x, y, θ) to the corresponding

[17]Strictly speaking, the free region in configuration space intersects all of the facets in the CS.

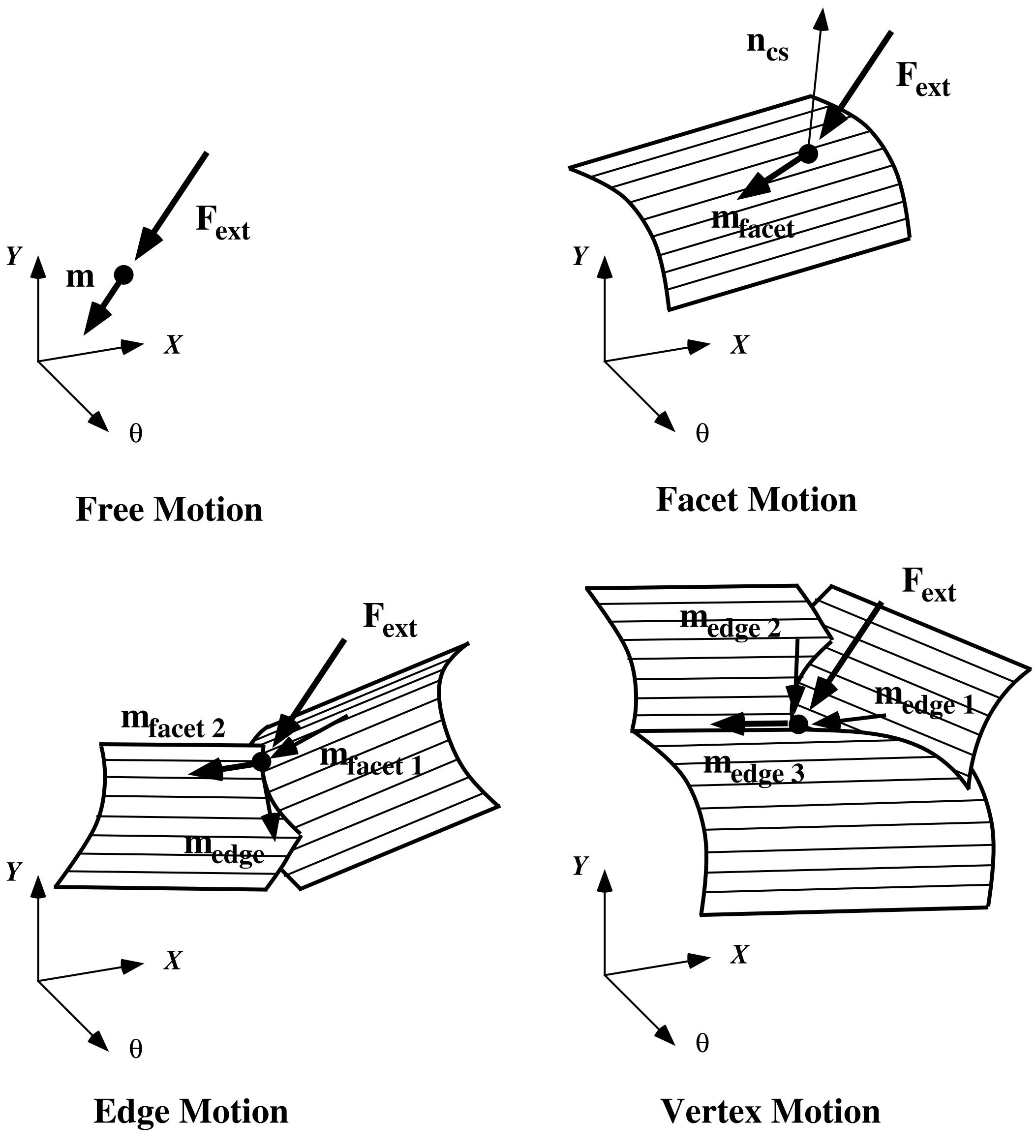


Figure 5.13: Selecting a consistent motion for *free*, *facet*, *edge*, and *vertex* contact states.

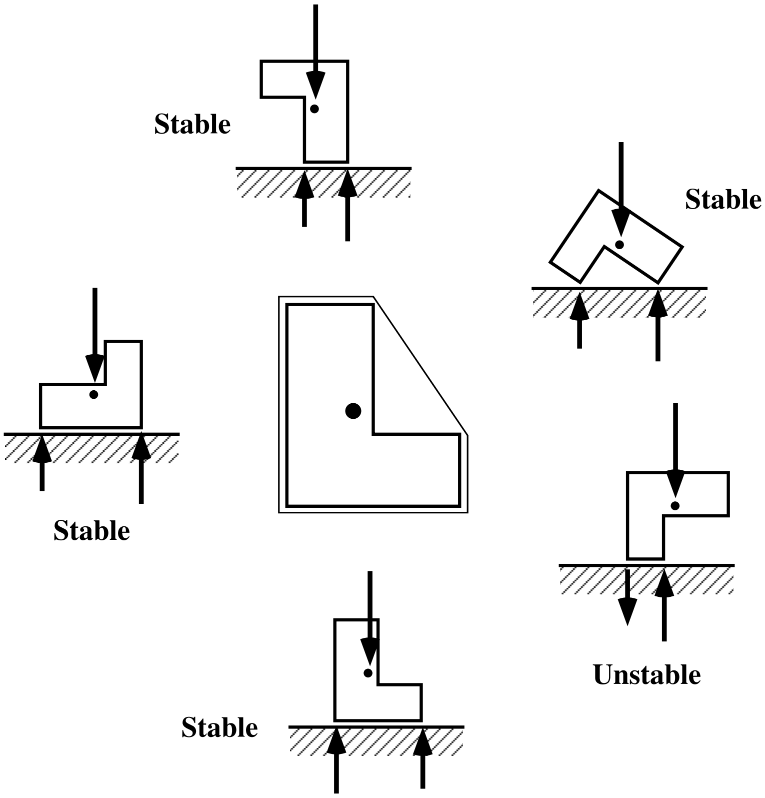


Figure 5.14: Constructing the initial contact state for a stable part orientation.

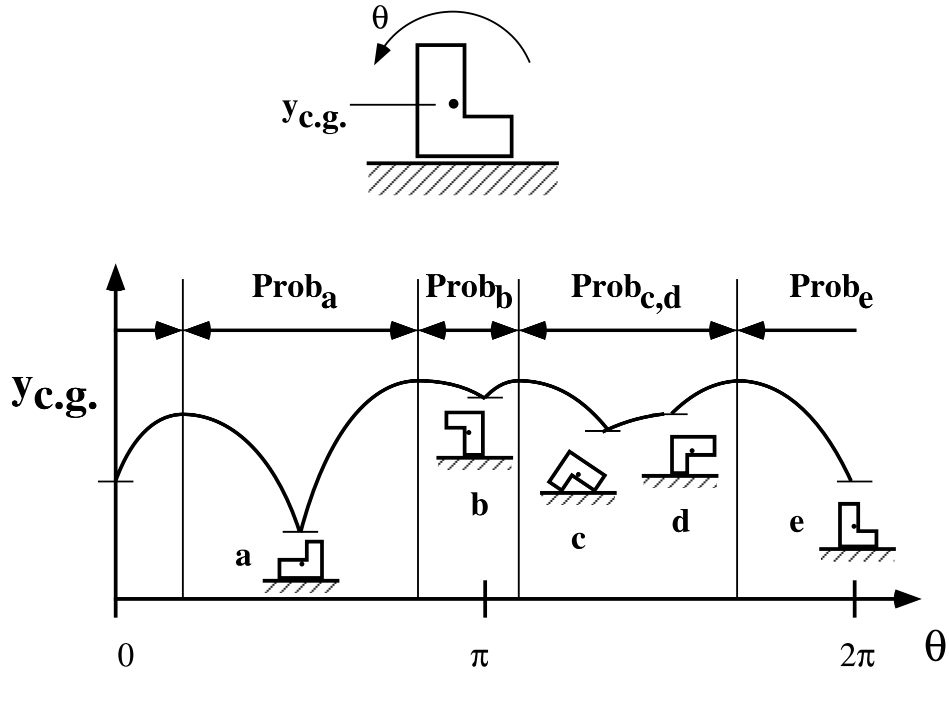


Figure 5.15: Determining the stable rest orientation probabilities for a part.

(p, θ) facet parameters and the facet contact state is initialized normally as described in earlier sections. The mouse based selection routine is described in Section 5.7.

5.4.2.4 Iterative Stepwise Integration

All of the motion computation operations discussed above are combined within an incremental motion path integration loop that is executed on command by the user.

For each initial contact state we execute the following procedure:

`path_integration_loop()`

- `compute_proximals()` Compute the array of proximals determining the relationship between the current position and the neighboring adjacent and intersecting facets.
- **Begin loop** Begin the incremental motion integration loop.
  - `test_motion_termination()` Check to see if current position is outside of the (x, y, θ) motion bounding box. If there is a support map check to see if the current position is supported.
  - `integrate_motions()` Integrate the incremental motion from the current position, including any sub-motions required to check for motion consistency.
  - `select_motion()` Select the consistent motion.

    If there is no motion, exit with **no_motion** flag.

    If the motion has left any existing contacts:
    - `create_contact_state()` Create a new contact state corresponding to the reduced set of contact facets.
    - `compute_proximals()` Recompute the array of proximals for the old position in the new contact state.
    - `record_and_increment_position()` Record the current position as the *first* position in the new contact state.
    - **current state = new state** Continue with the new contact state.
  - `compute_proximals()` Compute the array of proximals for the new position in the contact state.
  - `detect_new_contact()` Compare the old and new proximals to determine if a new contact has been made, and select a single new contact facet if necessary.

    If no new contact was made, then:
    - `record_and_increment_position()` Record the new position in the contact state.

Else, if a new contact was made:

`record_and_increment_position()` Record the new position as the *last* position in the current contact state.

`create_contact_state()` Create a new contact state corresponding to the larger set of contact facets.

`compute_proximals()` Recompute the array of proximals for the new position in the new contact state.

`record_and_increment_position()` Record the new position as the *first* position in the new contact state.

Go To **Begin loop**

End

Upon exiting, `path_integration_loop()` returns a flag indicating the mode of path termination which is one of:

- **no_motion** – Motion has stopped at a vertex or due to sticking.
- **loss_of_support** – The motion is in an unsupported region of the CS.
- **outside_motion_bbox** – The motion has left the allowable region of (x, y, θ) configuration space.
- **error + error_code** – An error identified by **error_code** has occurred within `path_integration_loop()`.

We have now described all of the major components required to construct motion paths in configuration space.

5.5 Computing Support Transitions

5.5.1 Assumptions

The following is a list of the assumptions made in computing the support status of points on the surface of the CS:

- The moving objects are assumed to be flat, fully planar polygons with negligible thickness.
- The reference point representing the center of gravity (**cg**) through which all external forces are applied to the part is assumed to lie within the contour of the part.

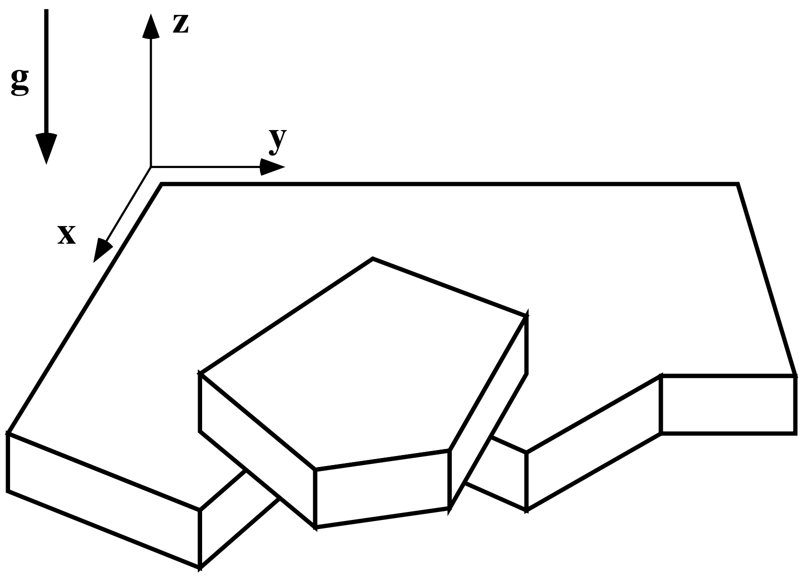


Figure 5.16: Determining the support of a polygon lying on top of another polygon.

- Although we model the applied force as a vector in the (x, y) plane applied to the part **cg**, we will implicitly assume there is always a small $(-z)$ component to the force vector pointing into the plane. It is this vertical force, for example due to gravity, that will cause parts in unsupported positions to fall off of the vibratory feeder track.

5.5.2 Determining the Support Status of a Planar Polygon

In the previous sections we have focused on computing force interactions between polygons in the plane where all reaction forces to an applied force occur at the boundaries of the polygons. In defining the support status of a polygon, we consider the case in which one polygon rests **on top** of another polygon, as shown in Figure 5.16. We define support to be a state of force equilibrium to a vertical component of an applied force at the **cg** where all of the reaction forces between the top and bottom polygons are positive, i.e. all contact forces push against rather than pull on the other object to maintain equilibrium. Figure 5.17 shows a number of examples of polygons in both stable and unstable orientations.

We may determine the stability of a polygon's support geometrically by examining the set of points at which the top and bottom polygons are in contact. Each point in the set formed by the intersection of the two polygons is the site of a potential reaction force that will contribute to the force equilibrium of the top polygon. If we

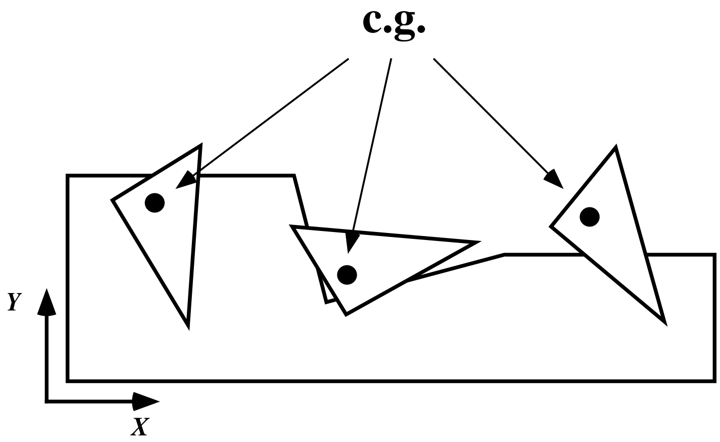


Figure 5.17: Examples of stable and unstable support configurations.

draw a line in the (x, y) plane through the **cg** point of the top polygon, then the polygon will be in static equilibrium in rotation about that line if there exist contact points on both sides of the line through which reaction forces may act to produce a zero net moment. If we are able to find a pair of points on both sides of a line at any orientation in the plane passing through the **cg**, then the polygon is in static equilibrium for all possible rotations out of the plane. This immediately suggests a method for testing the stability of one polygon resting on another. Specifically, given two polygons $Poly_1$ and $Poly_2$, with $Poly_1$ resting on top of some part of $Poly_2$, we may say that $Poly_1$ is supported by $Poly_2$ iff the **cg** of $Poly_1$ is within $CHull[Poly_1 \cap Poly_2]$, where $CHull[]$ is the convex hull operation for a set of points in the plane. More formally, we have:

$$Int[Poly] = \bigcap l_i,\ i = 0, 1, 2...$$

$$\mathbf{cg}_{Poly_1} \in Int\,[Chull[Poly_1 \cap Poly_2]]$$

where $Int[Poly]$ is defined as the set of points forming the *Interior* of a $Poly$ formed by i half-planes bounded by lines l_i.

Because the interior of a convex hull is always to one side of every edge on the hull, any line through a point in the interior of the convex hull will intersect the hull at two points on either side of the interior point. Looking at Figure 5.18 we see that this test gives the intuitively correct result for each of the examples in Figure 5.17.

The above test for determining the support status of a polygon has the appeal that it is conceptually simple. Unfortunately, computing $Poly_1 \cap Poly_2$ requires that we generate and test $O(n^2)$ line intersections where n is the number of vertices or edges in each polygon, and $CHull[]$ requires $O(mlogm)$ computation where m is the

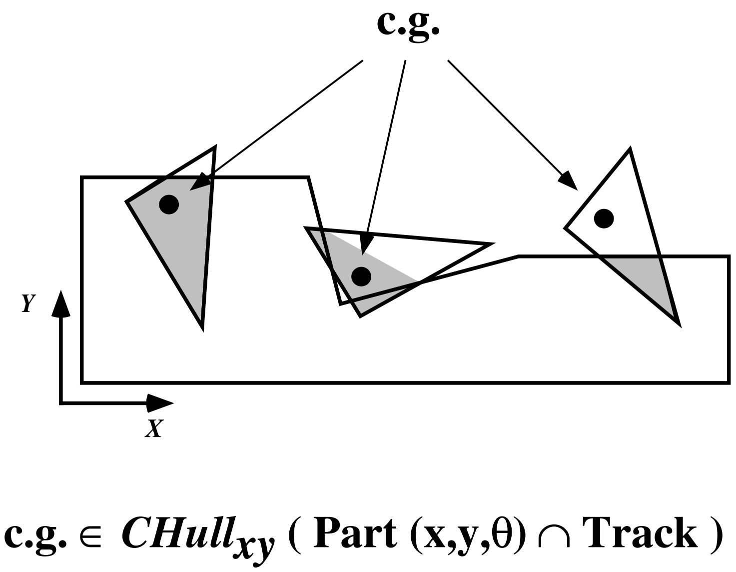


Figure 5.18: Stability test for polygons using $CHull[Poly_1 \cap Poly_2]$.

number of vertices in the intersection [64]. It turns out that we can do better than this by realizing that we do not actually need to perform the sorting and ordering of the intersection vertices required to construct the $CHull$. In fact, depending on the particular case being tested we may not even need to construct all of the vertices of $Poly_1 \cap Poly_2$.

For any collection of points P, any subset S of P is contained within the $Chull[P]$, and therefore $Chull[S]$ is contained within $Chull[P]$. This is useful because if we can determine the **cg** to be within the $CHull$ of a subset of support points, then we know that the **cg** is supported without having to examine all possible support points. In particular, if we determine that the **cg** is not *extremal* to a subset of support points, then the part is supported.[18] On the other hand, if the **cg** is found to be extremal to the complete set of support points, then the part is unsupported. To determine if a **cg** point is extremal to a set of points that are computed one at a time we perform the following steps:

1. Given the **cg** point and two support points, compute a pair of vectors, each pointing from the **cg** point to one of the two support points.

2. Label the vectors *left* and *right* such that $\vec{r}_{right} \times \vec{r}_{left} > 0$.

3. For each new point p, compute the vector $\vec{r}_p$ and determine if it is to the left of

[18]An extremal point is a point on the convex hull of a set of points. See Preparata and Shamos [64].

$\vec{r}_{left}$ or to the right of $\vec{r}_{right}$ by taking the cross products of the respective vectors with $\vec{r}_p$ and checking the sign of the result. If a sign change occurs, replace the corresponding left or right vector with $\vec{r}_p$, and recompute $\vec{r}_{right} \times \vec{r}_{left}$.

4. If the cross product changes sign, i.e. becomes negative, then the **cg** point is no longer extremal to the set of points tested, and the part is **supported**.

5. Else, compute the next support point p and go to step 3.

6. If there are no more support points to test, and $\vec{r}_{right} \times \vec{r}_{left} > 0$, then the **cg** point is extremal to the set of support points and the part is **unsupported**.

This process is illustrated in Figure 5.19.

By testing each point in $Poly_1 \cap Poly_2$ as it is generated, we improve the average performance of the overall support test because as soon as we generate a point that places the **cg** in the interior of the existing set of intersection points we can stop and report the part to be supported. In the worst case, which will occur when the **cg** is in fact unsupported, our test is still $O(n^2)$ because we will have to generate and test every point in $Poly_1 \cap Poly_2$ since we never know when the next point in the intersection might make the **cg** point no longer extremal.

5.5.3 Approximations and Optimizations

On average the extremal point test provides a considerable improvement in computing the support status of a point in configuration space. Nevertheless, computing support regions by simply testing and labeling as supported or unsupported discrete points would be unacceptably slow for an interactive design environment. In this section we discuss a series of hierarchical tests and optimizations that can reduce even further the amount of necessary computation.

To begin with, we will compute the support status of only those configuration space points that lie on the surface of facets in the CS, and as was the case for motion computation we will not explicitly represent edges or vertices on the CS. Another simplification arises from the observation that, assuming the **cg** is contained within the contour of the part, the part will be supported if the **cg** point lies within the interior of the support polygon. Similarly, if the **cg** point lies outside of the $CHull[]$ of the supporting polygon, then no support is possible since there can be no support points outside the support polygon's $CHull[]$. For a given position of a part's **cg** we can compute the *state* of the **cg** point with respect to the support polygon and its convex hull. Table 5.2 lists the possible cg states and their resulting support status:

The main advantage of this test is that we need only compute the status of a point with respect to the support polygon, as opposed to computing the intersection of both the part polygon and support polygon. Since a facet surface represents the

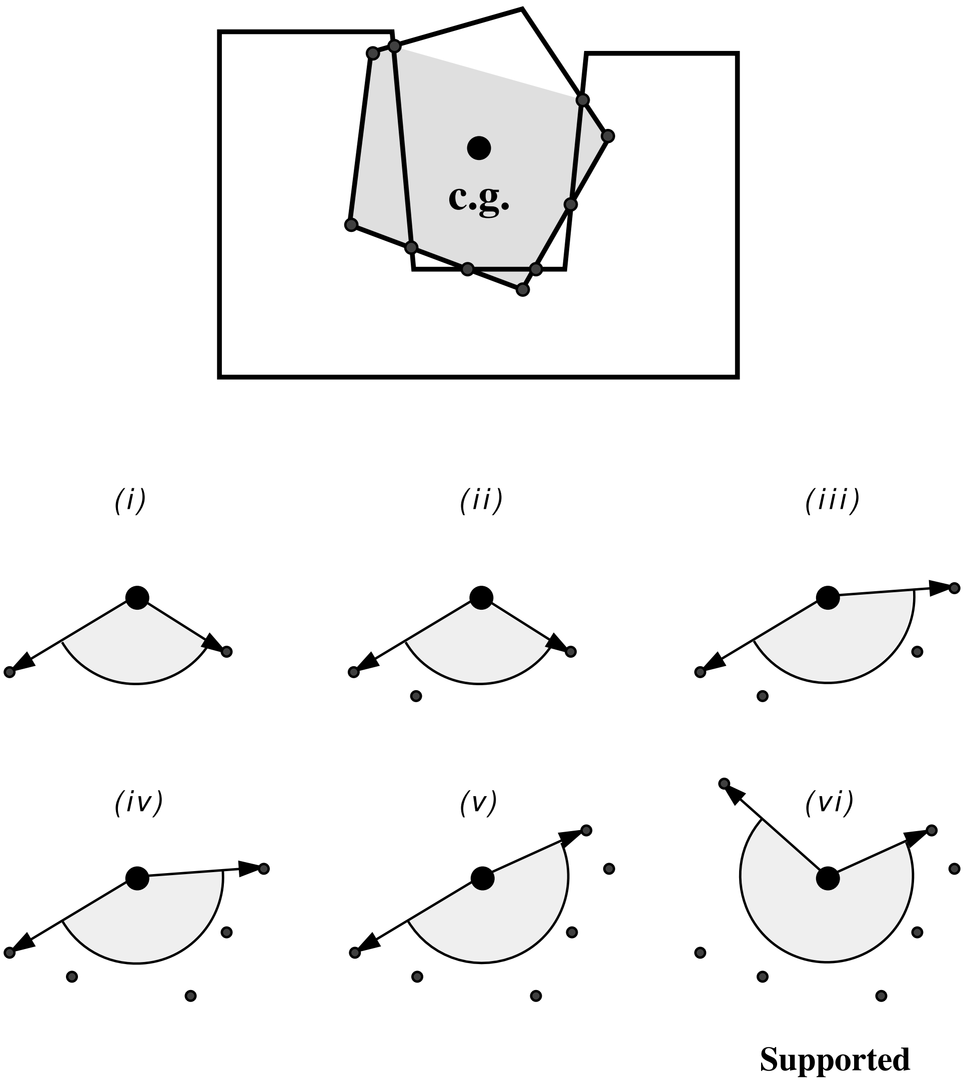


Figure 5.19: Testing the **cg** point against a series of points to determine if the **cg** is *extremal.*

Support Status from Point_In_Poly? Tests for Part cg.		
cg in Support_Poly?	*cg in CHull[Support_Poly]?*	*Part Support Status*
Point Outside	*Point Outside*	Unsupported
Point Outside	*Point Inside*	Test Support
Point Inside	*Point Outside*	–
Point Inside	*Point Inside*	Supported

Table 5.2: Table of part support status as determined by **Point_In_Poly?** status of the part **cg** w.r.t. the Support_Poly and CHull[Support_Poly].

set of possible positions of the part **cg** for a given contact, the **Point_In_Poly?** test also allows us to hierarchically test for support starting first with an entire facet, and then smaller subregions within a facet determined by ranges of (p, θ). Only in those cases where the result is **Test Support** will we need more extensive and detailed testing using the extremal point routine.

Testing a Whole Facet's Support Status:

We first determine if an entire facet may be supported or unsupported by checking the facet's bounding box against both the **Support_Poly** and the **CHull[Support_Poly]**.[19] If the bounding box is inside of the **Support_Poly** or outside of the **CHull[Support_Poly]** then the part is supported or unsupported, respectively, over the range of positions determined by the facet's surface. If the bounding box intersects either of the polygons, or is between the **Support_Poly** and the **CHull[Support_Poly]**, then the facet's support status must be examined in more detail.

Testing Regions of a Facet:

For those facets that intersect either the **Support_Poly** or the **CHull[Support_Poly]**, we must subdivide the surface of the facet into regions of differing support status. For simplicity, we choose to divide the facet into strips of fixed resolution that will be tested individually. To do this more efficiently, we first cache the intersecting polygon segments from the bounding box test for later use to reduce the amount of subsequent testing required. Then, based on the facet's size and curvature as given by l (the length of the polygon edge forming the contact facet) and r_c (the distance from the **cg** of the moving polygon to the point of contact), we determine the maximum Δp and $\Delta\theta$ increments that are within the RES_{xy} specified for computing the support status over a facet's surface. Using this information we divide the facet into a series of (x, y) slices at increments of $\Delta\theta$ within $[\theta_{min}, \theta_{max}]$. Each of these slices

[19] We compute the (x, y) bounding box of the facet over the range $[\theta_{min}, \theta_{max}]$.

corresponds to a line segment in (x, y), and represents a translation of the part **cg** along a contact edge at a fixed orientation.

Testing Along a Facet Slice:

For each $\Delta\theta$ increment, we generate the facet slice line segment and test it for intersections with the polygon segments from **Support_Poly** and **CHull[Support_Poly**] that were cached in the facet's bounding box test. All intersections are recorded and sorted by the value of the parameter p at which they intersect the facet segment. The **Point_In_Poly?** status of the facet segment's start point ($p = 0$) is then computed, and the segment is divided into **(Supported, Unsupported, Test Support)** ranges by appropriately toggling the point status from $p = 0$ at each intersection with the segment from $p \in [0, 1]$ and labeling each following range accordingly. Figure 5.20 illustrates this labeling process for a typical facet slice. Each segment range labeled **Test Support** must be explicitly tested at Δp increments using the extremal point test discussed earlier, and neighboring incremental test points with the same resulting support status are then merged. The final result is a facet slice line segment divided into a series of consecutive **Supported** or **Unsupported** sub-ranges, whose adjacency points correspond to support transitions.

Each facet slice containing a support transition is recorded as an element in an array of slices indexed by θ_{slice}. Each slice element in turn contains the support status of the ($p = 0$) point of the slice, and an array of support transition flags (**Lose Support, Gain Support**) indexed by $p_{transition}$ that indicate the type of support transition. This array of arrays, illustrated in Figure 5.21, is stored in the facet's *support_map* field. The support_map is used both by the rendering routines to display the supported and unsupported regions of the facet, as well as by the motion computation routines to determine if a path has entered a ref_facet's unsupported region. Specifically, for each point generated along a motion path, we check the corresponding (p, θ) position on the ref_facet against the support_map to determine if that point is unsupported and should therefore be terminated.[20]

5.6 Rendering and Display

Each facet surface is rendered in the (x, y, θ) configuration space as a series of polygons, each bounded by four vertices. Using a constant predetermined $\Delta\theta$, the facet is divided into equally spaced slices from $\theta = [\theta_{min}, \theta_{max}]$, each of which is bounded by a pair of (x, y, θ) vertices computed from the facet equations at $p = 0$ and $p = 1$. Each polygon structure points to two consecutive pairs of vertices, along with their

[20]For those portions of a path that are in *free* configuration space, we must test each incremental point along the path explicitly using the extremal point test since no support_map is available.

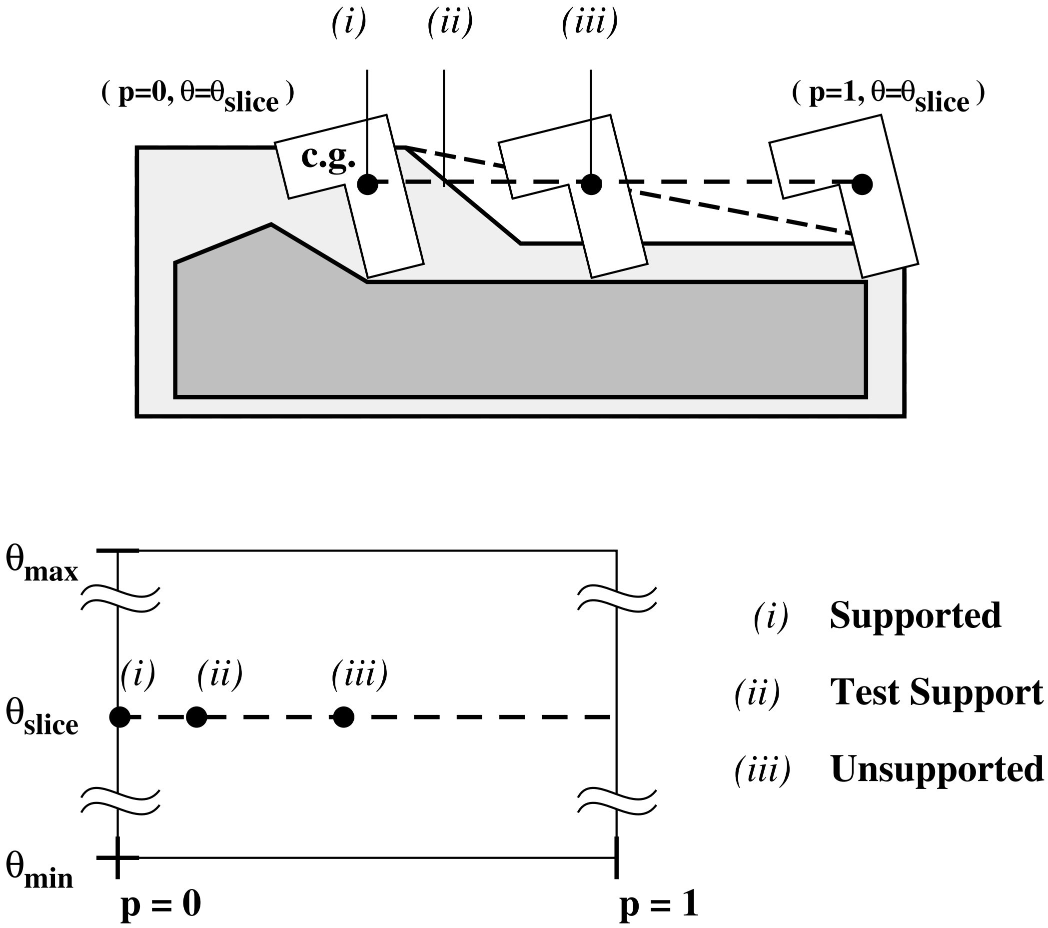


Figure 5.20: Subdividing a facet slice into differing support state ranges.

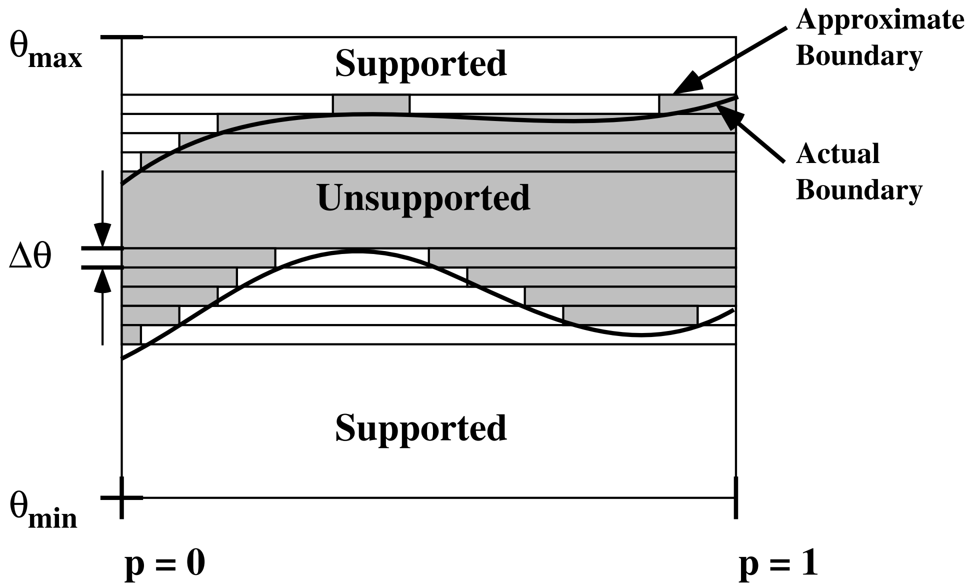


Figure 5.21: A typical support map for a facet in terms of (p, θ).

corresponding facet normal vectors, arranged counterclockwise around the outward facet normal. In addition to the vertices and normals, each polygon structure contains indices to the edge and vertex of the moving and stationary object polygons that form the contact facet, as well as indices to the palette of colors and material lighting properties used to render the polygon. When enabled by the user, each polygon is drawn with a dark boundary line to highlight the facet contour. Otherwise, the polygons making up each facet are simply drawn and blended using the Phong shading model in the Silicon Graphics *GL* rendering library.

Each complete *forward* facet is rendered individually.[21] Those facets or portions of facets that are occluded in the complete CS set are hidden from view by the depth buffer, or Z-buffer, of the graphics workstation. When motion paths are computed in configuration space, they are rendered as curves composed of line segments connecting the points along the path, and are drawn slightly above the surface of the CS facets. The line thickness of each path is drawn proportional to the probability of the initial position for that path so that the more common part motion paths are thicker than the less likely ones.

[21] *Backward* facets are by definition interior to the set of CS facets, and hence will be hidden from all viewing angles.

5.7 Interactive Design Functions

With the exception of specifying file names via the keyboard, all user inputs to `cspace-shell` are made via the mouse. Vertices of the object polygons (part, bowl wall, and track) may be selected directly within the object display windows or indirectly from the CS in the configuration space display window. Non-shape parameters, such as the coefficient of friction μ or the free-space start position for the computation of a single motion path, are modified using simple linear sliders.

When the user selects a CS feature by clicking the mouse in the configuration space display window, the X-Y position of the cursor in the window is transformed into a line in the configuration space coordinate system corresponding to the current view using the Silicon Graphics `mapw` library function. This line is then tested for intersections with every facet rendering polygon in the CS that is visible, i.e. facing the user. The intersected facet rendering polygon that is closest to the viewer is determined, and the facet type, index to the nearest rendering polygon vertex, and the approximate (x, y, θ) position selected on the facet surface is recorded.

Parametric Design Functions:

The information obtained from the selected vertex of the rendering polygon in the CS may be used to identify either the contacting object vertex or edge vertices that form the corresponding selected facet. We determine which feature will be selected by *a priori* assigning edge vertices of the moving object polygon to type A facets and edge vertices of the stationary polygon to type B facets. Therefore, the index number returned by the selected vertex of the rendering polygon may be used to select the corresponding vertex of an object polygon. To make this coupling apparent to the user, all of the facets in the CS that are affected the selected object vertex are highlighted along with the vertex itself. This is accomplished by changing the color palette index of every rendering polygon in the CS that corresponds to the selected object vertex.

The user may move all of the currently selected object vertices together by selecting a point in configuration space and moving away from it with the right mouse button held down. In this case, an (x, y) plane is displayed in the configuration space at the θ value where the user initially clicked on the CS. Movements of the mouse from this point are translated into incremental (x, y) motions by intersecting this plane with the `mapw` line from the new mouse position. The (x, y) motions are applied directly to all selected vertices of the stationary polygon, and inversely via an inverse rotation matrix at the (x, y) plane's θ value, i.e. $Rot[\theta]^{-1}$, to all selected vertices of the moving polygon. The result is a change to the CS surface that tracks the motion of the mouse in the (x, y) plane from the initial click point, along with the modifications to the object vertices (shape) necessary to make this change to the

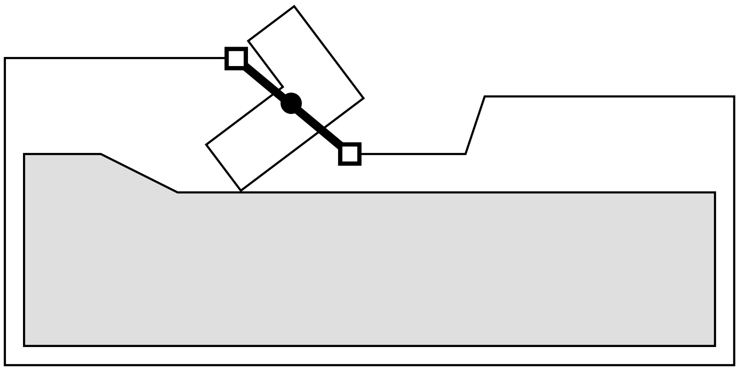


Figure 5.22: Critical track vertices for a part near a support transition.

CS.

In another mode of operation the user may elect to change the shape of the track polygon by a movement of the mouse, instead of changing the shapes of the moving or stationary polygons, by clicking on or near the boundary of a LOS region. In this mode, the (x, y, θ) facet position that the user clicked on is used to determine the track polygon vertices that are critical to the support transition at or near that position in configuration space. Critical vertices are those vertices that define segments on the track whose intersection points with the part boundary are nearly along a line through the **cg** point of the part that marks the transition from the **cg** being extremal to non-extremal, as shown in Figure 5.22. The selected track vertices are highlighted, but unlike the CS selection operation the LOS regions that are coupled to these track parameters are not highlighted.

The user modifies all of the currently selected track vertices by selecting a point on the CS near a LOS boundary and moving the point away from it while holding down the right mouse button. Differential motions in (x, y) are computed by again projecting the `mapw` line from the screen position of the mouse into configuration space and finding the closest point on a line collinear to the facet slice at the θ value of the original click point. This offset motion is applied directly to the currently selected track vertices, and the result is a motion of the LOS boundary that nearly tracks the motion of the mouse. Due to the extremely nonlinear nature of part/track intersections that create the LOS boundaries, the movement of the LOS boundary tracks that of the mouse only for relatively small excursions away from the initial click point. The mapping from mouse movements along a facet slice is analogous to moving along a tangent to a point on a nonlinear curve; the further we move away from the point of tangency the worse the approximation becomes.

Non-Parametric Design Functions:

The interactive functions described above all allow variational modification only of existing design parameters. We may also introduce entirely new contours to the track profile by generating a cutout from the track. The user selects a cutout by specifying an (x, y, θ) position from the CS and an oriented line in the (x, y) plane of the track about which a part should rotate out of the plane to fall off the track. The motion of the part rotating out of the plane about this line produces a cutout contour in the track corresponding to the shape of the part.[22] `cspace_shell` implements an approximation to the cutout operation in which the generated cutout contour is the convex hull of the actual contour that would be cut from the track as the part was rotated out of the plane about the oriented line.

Using the selected (x, y, θ) position and oriented (x, y) line, the cutout routine generates a polygonal contour around the intersection of the (x, y, θ) translated and rotated part contour and the half space to the right of the oriented line passing through the part **cg**. The convex hull of this contour is taken and expanded in (x, y) by the amount RES_{xy} that is currently specified for computing the support status over a facet's surface. The ordering of the expanded contour's vertices are then inverted to form a hole, and the polygon is intersected with the track polygon at the selected (x, y) position. If the cutout contour intersects the track boundary, it is cut and it's vertices spliced into the track polygon. If the cutout contour is completely within the track polygon, the cutout vertex closest to the track edge is split and connected to the track polygon via a narrow slot. The result in either case is a new track polygon contour of genus zero. The cutout operation is illustrated in Figure 5.23.

5.8 Summary

Based on the assumption that `cspace_shell` would be used interactively on problems of small to moderate size, i.e. polygons with $O(10)$ edges/vertices, our implementation strategy focused on reducing the *average* running time required to compute motion constraints for such problems versus developing algorithms aimed at reducing the asymptotic running time of larger problems.

5.8.1 Optimizations

Some of the major optimizations to improve the performance of `cspace_shell` and its subroutines that were discussed earlier are summarized here for reference.

[22] This contour is actually an approximation to the cross-section of the volume swept out by the part as it rotates out of the plane to fall off of the track.

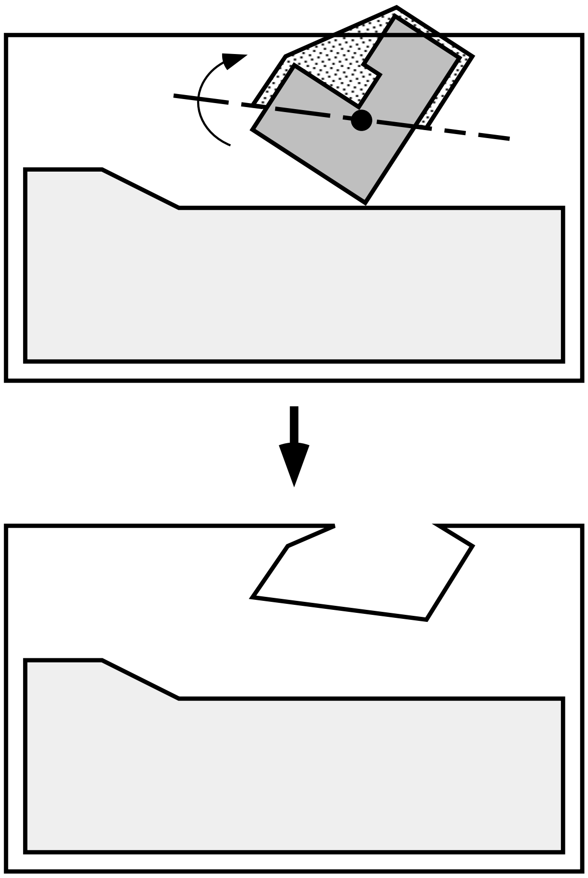


Figure 5.23: Convex approximation to a track cutout formed by a part falling off of the track.

Generation and display of motion constraints:

- Compute and display only the CS facets during shape modification. Other CS features such as edges, vertices, and facets or portions of facets that are occluded are left implicit.
- Use depth-buffering to hide occluded surfaces in the CS display.

Computation of paths and topology:

- Compute local CS topology only during motion path integration.
- Cache information on facet intersections and adjacencies in during path computation for use in subsequent path computations.

Computation of facet support maps:

- Test for support status using a hierarchy of tests from entire facets to sub-facet regions.
- Divide facets into regions of differing Point_In_Poly status of the part **cg** relative to the track and $CHull[]$ track polygons.
- Cache intersecting facet/track segments for use in repetitive support computation and testing.

5.8.2 Complexity and Performance

Computational Complexity

The following are estimates of the *worst case* complexity for the three main processes running in `cspace-shell`: computation of contact facets, computation of support regions (support maps) on the facets, and numerical integration of the motion paths (including the CS topology computation). In actual usage, the optimizations listed above combined with the inherent structure of the representations tend to make computations more tractable, as shown below. Each of the estimates below assumes that all polygons have $O(n)$ vertices.

Contact Facet Generation

Computation of a contact facet requires *constant time* and consists of computing a few constants and placing them into a data structure. A contact facet is generated for each *edge-vertex* and *vertex-edge* contact between two polygons, producing $2n^2$ contact facets. The time complexity in terms of *polygon size* n is therefore $O(n^2)$.

Support Map Computation

Computing the support map for a contact facet, in the worst possible case, consists of testing whether or not the moving polygon's **cg** is *extremal* to the set of possible support points at the given (x, y, θ) position of the moving polygon. This set consists of the intersection of the moving and supporting polygons at that position. For two arbitrary polygons there will be $(2K_1 n + K_2 n \log n)$ vertices to be tested, where K_1 and K_2 are constants. Points are sampled across the facet at $N = \left(l_{edge} \frac{1}{RES_{xy}}\right)$ intervals along $M = \left(r_{contact}(\theta_{max} - \theta_{min}) \frac{1}{RES_{xy}}\right)$ fixed-θ slices of the facet. Therefore, the required computation is $(2n^2(2K_1 n + K_2 n \log n)NM)$. In terms of *polygon size* n we have a time complexity of $O(n^3 \log n)$, and in terms of *resolution* RES_{xy} we have $O\left(\frac{1}{RES_{xy}^2}\right)$.

Motion Path Integration

Computing motion paths consists of first determining and recording those facets that might be encountered through adjacent transitions or intersections with the current reference facet, which is constant for computing adjacencies and requires $K(2n^2)^2$ testing for facet intersections, where K is a constant. For each step in the integration we must check for transitions relative to each of the $C(2n^2)$ facets listed from the previous computation, where $C < 1$. Integration steps are made at maximum intervals of the resolution given by RES_{xy}. The actual motion computation at each step requires constant time T_{comp}. Therefore, the computation required consists of $\left(K(2n^2)^2 + P_{count} \frac{1}{RES_{xy}} C(2n^2)\right)$, which gives a worst case time complexity of $O(n^4)$ in terms of *polygon size* n, and in terms of *resolution* RES_{xy} we have $O\left(\frac{1}{RES_{xy}}\right)$.

Average Performance

The above complexity expressions are worst case bounds. The average time required to compute and render constraint facets for polygons on the order of $8 \approx 10$ edges was ≈ 0.6 seconds. The average time required to compute and render support maps for the same polygons was $2 \approx 5$ seconds for a resolution of 0.1 inches. Finally, the time required to compute motion paths for the same examples was $3 \approx 7$ seconds for an average of 4 paths.

Conclusion

Chapter 6

6.1 Summary

The primary goal of this research was to argue for the validity of the hypothesis that motion constraints can serve both as a representation of the function that may be derived from shape interactions and as a tool for manipulating and designing such functional shapes.

We identified and explored two key issues raised by the above hypothesis: the development of representations for function in terms of motion constraints, and the development of tools and methodologies needed to create functional shapes from motion constraints.

We developed a formal representation of kinematic motion constraints for a simple class of geometric contacts (planar polygons) and non-kinematic constraints including forward projections of motions for both exact and energy-bounded dynamic models. These representations were applied to the modeling of examples chosen from four application domains: compliant assembly of rigid objects, orienting of parts by vibratory bowl feeders and APOS vibratory feeders, and fixtures used to locate and hold parts. These examples were used both to test the validity of the hypothesis, and to inspire and guide the development of the detailed representations. For the purposes of visualization, the abstraction of function via functional metaphors within each of the application domains was a particularly powerful tool in interpreting the representations and determining what changes were necessary to achieve the desired function.

We developed a series of tools and techniques to access and manipulate these representations for the purposes of design. The basic approach was to construct parametric representations of motion constraints for planar polygons, and provide access to these representations in an interactive graphical environment that allows the manipulation of shape parameters in the context of function represented as motion constraints.

The combined representations and tools were applied to the design of specific examples from two of the four domains: vibratory bowl feeders and compliant assembly systems.

Some of the more important ideas developed in the research include:

- **Unified Representation and Design Environment.** The tasks of analysis, verification, visualization, *and* design of function from shape may all performed in the context of motion constraints using a common set of representations.

- **Explicit Representation of Motion Constraints.** An explicit representation of shape *interactions* focuses the designer's attention on what matters – motion constraints. Specifically, we make use of computational power to make explicit what is usually implicit (i.e. motion constraints from shape), and thereby direct the designer's attention to the important functional attributes of a design. Functional metaphors help to put these constraints in the proper context for visualization and reasoning.

- **Rapid Computation of Constraints.** Representations for motion constraints may be computed and displayed *quickly* enough to be used interactively for analysis and design.

- **Dynamic Constraint Visualization.** Real-time display and interactive manipulation of motion constraints exposes coupling among constraints and sensitivity to parametric variations in ways that would otherwise be difficult to visualize. By making explicit the coupling inherent in motion constraints we expose the basic limitations of what is and what is not possible to achieve consistently by means of shape modifications. Simply stated, the representations don't lie, nor will the tools allow us to cheat or do the wrong thing. This is in direct contrast to techniques that may produce inconsistent constraints generated from swept shapes.

- **Simulation Supersets.** Forward projections and kinematic motion constraints act as a kind of *superset* of simulations that allows one to examine all possible motions at once for a given system, including motions that may not be simulated in detail.

- **Simplified Representations Control Complexity.** Some properties of inherently complex phenomenon may be captured with simpler representations. Examples included complex dynamic interactions represented as energy bounded forward projections, and transitions to higher dimensional motions represented as transitions from supported to unsupported configurations in a lower dimensional configuration space.

- **Foundations for Design Automation.** Mathematically precise and computationally accessible representations for motion constraints were developed that help lay the groundwork for subsequent semi and fully automated design approaches.

At the core of the research is an implemented computational environment for the analysis, visualization and design of motion constraints among systems modeled as planar polygons – `cspace-shell`. Specifically, `cspace-shell` supports:

- the display of motion constraints, both contacting and for planar support,
- the ability to manipulate shape parameters directly via motion constraints that permits exploration of coupling between design parameters and constraints,
- the simulation and animation of system behavior for specified inputs and initial conditions, and
- the ability to directly generate shapes and constraints for a limited set of imposed motions via a cutout operation.

6.2 Discussion

We return to the question posed in Section 4.6, following the design examples. *How useful were the representations and tools for design?* We will discuss first the underlying concepts and methodologies, and then turn our attention to the implementation of `cspace-shell`.

6.2.1 The Concepts

In Section 1.2.3 we outlined the major goals of this research. The primary goal was to be able to reason about and create functional shape interactions, from which we derived two major sub-goals: *(1)* develop a precise and accessible representation for functional shapes using motion constraints, and *(2)* develop the tools and methodology necessary to manipulate motion constraints for design. Chapter 2 described a set of motion constraint representations, for objects modeled as planar polygons, that are *mathematically precise* and *computationally accessible*. Chapter 4 described a set of functional shape synthesis procedures and tools for the representations that are *functionally consistent* and *computationally tractable*, and applied these tools to specific examples from a number of application domains.

We proposed that the representations and synthesis tools developed in the research should be applicable to a well defined set of examples. To ensure this, Chapter 3 introduced four application domains chosen from the field of automated assembly. The representations for these four example domains in Chapter 3, the design examples for two of the four domains in Sections 4.4.6 and 4.5.3, and the experimental verification of feeder designs in Section 4.4.7 suggest that these criteria have been satisfied, at least for the domains so far examined. In terms of the overall usefulness and potential of motion constraints as a tool for analysis and design of functional shapes, the general approach and implemented tools demonstrated both strengths and weaknesses, some of which we will now discuss.

Strengths

Most of the perceived strengths of the motion constraint representation have been discussed extensively elsewhere in this report. Briefly, the kinematic and non-kinematic motion constraint representations capture explicitly what we care about in terms of function from shape. In terms of interactive design, the ability to reason about function *and* manipulate design parameters in the context of motion constraints allows us to determine at a glance whether or not a particular design will work. Furthermore, the dynamic constraint visualization provided by the interactive environment will indicate what parameters must be changed if the existing design doesn't work by exposing the inherent properties of motion constraints.

Weaknesses

We noted in the introduction to the configuration space representation in Section 2.2 that the complexity associated with representing motion constraints grows exponentially with the number of degrees of freedom. We have, in the example domains examined, been able to limit the number of degrees of freedom that must be considered explicitly by judicious choice of models and assumptions. Unfortunately, there will no doubt be numerous examples in which such simplifications will not be suitable. In such cases, the growth in complexity could render suitable motion constraint representations computationally intractable.

In the domain of compliant assembly introduced in Section 3.1 and evaluated in Section 4.5, we noted that in addition to the functional constraints that determine the assemblability of a set of parts, there are typically other non-assembly factors that must be taken into consideration when determining what modifications to shape may be made. In the examples presented, we assumed that whatever geometric properties were critical to such non-assembly functionality would not be modified. As we noted, we would ideally want to consider such properties *together* with motion constraints. The problem of integrating the representation of motion constraints with other functional considerations remains open.

In terms of using motion constraints as a tool for interactive shape design, we are faced with the limitation of having to represent constraints in three dimensions or less. In general this means that at any one time we may consider in detail only those motions that possess three degrees of freedom or less.[1] Although we were able to derive models for the four application domains that captured essential motions in planes with three degrees of freedom, and in some cases were able to make due with only representations of *transitions* to higher dimensional motions, we may not always be so fortunate. Those cases where the important functional motions have more than three degrees of freedom will in general be beyond the scope of interactive design tools.

Finally, and perhaps most importantly, there is the concern that the representation of motion constraints in configuration space may not be the most intuitive representation available in which to consider shape interactions. In particular, because the representation presents the set of *all* motion constraints together there is the risk that the designer may be overwhelmed with excessive detail. Indeed, there is a great deal of information presented in an image such as Figure 3.12 in Section 3.2. We have tried to soften the blow somewhat by providing abstractions in the form of functional metaphors that focus ones attention on the more important aspects of the motion constraint representation for a given application domain.

Would it be possible to present a more abridged representation, perhaps along the lines of functional metaphor illustration such as in Figure 3.3 for a compliant assembly or Figure 3.9 for a bowl feeder? Such a simplified representation could certainly be made to indicate if a given design exhibited the desired behavior. For example, if more than one motion path arrow in the bowl feeder metaphor in Figure 3.9 passed through the feeder without being rejected, we would know that the given feeder design will fail. However, such a representation would not be of much use in determining what to do next, i.e. how to change the underlying geometry and dynamics so as to make the feeder function as desired. For such a task, we need to have access to the underlying constraint representations in order to identify and make the necessary changes.

Finally, unfamiliar representations are rarely intuitive at first glance. Much the same could be said of many engineering representations in widespread use today. For example, the information presented in such representations as a frequency domain Bode plot, a root locus diagram, or a phase plane portrait for a system from the fields of system dynamics and controls are arguably nonintuitive when first encountered. With experience, however, such representations become powerful visualization tools that provide valuable information in a concise and, eventually, intuitive manner. Similarly, the motion constraint representation arguably has the potential to be a useful tool when objectively judged in the context of what makes a good representation.

[1] As noted in Section 2.2 the three degree of freedom constraint does not prevent us from considering three dimensional shapes, although the present implementation makes use of planar models for computational efficiency.

6.2.2 The Implementation

The `cspace-shell` implementation, as noted above, presents a set of tools and synthesis procedures that are both functionally consistent and computationally tractable, as well as being sufficiently fast to be used interactively. Basically, the set of implemented representations and tools allowed us to do what we wanted, although not always as easily as we might have liked. Specifically, the discussion regarding the effectiveness of the design functions implemented in `cspace-shell` in Section 4.6 indicated that, although sufficient to generate workable designs, there was a good deal of room for improvement in terms of their ease of use.

The following is a list of features and operations implemented in `cspace-shell` **(in bold faced text)**, together with the features we would have *liked* to have had available while doing design.

- **Independent Manipulation of Contact Constraints OR Support Constraints.** The locations of support transition boundaries on the CS may be modified directly by apparent inversion of the support track geometry, or indirectly by modifications to the CS itself. But the indirect modifications that result from changes to the CS often make it difficult to position support transition boundaries as desired. One option to overcome this would be to introduce a new apparent inversion design function that couples support transition boundary modifications to *both* the track and part/bowl wall geometries to allow more uniform manipulation of these boundaries. Such a function would serve as an alternative to the present technique of having to alternately modify the CS and support transition boundaries independently of one another within the inner loop of the design methodology illustrated in Figure 4.19.

- **Representing and Manipulating Only the Intersection of the Full $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$ Support Boundaries with the CS.** The coupling between design parameters and motion constraints exhibited in the feeder examples of Chapter 4 complicated the process of design and accounted for a considerable amount of our time and effort, both in developing design methodologies to control complexity and in requiring many passes through the inner design loop of Figure 4.19. Although most of this coupling is an inherent property of motion constraints, some additional coupling was introduced by the fact that we were not able to manipulate support constraints independently of the contact constraints on the CS. Therefore, a more desirable representation than showing the support transition boundaries only where they intersect the CS would be the direct computation, rendering, and manipulation of the full support boundaries in (x, y, θ) configuration space together with the CS. This would be an alternative to the coupled manipulation of support transition boundaries proposed above.

- **Applying Modifications Uniformly to All Selected Object Vertices.** The apparent inversion design functions for manipulating the CS facets map to all selected polygon vertices uniformly. As a result, changing the shape of a facet to, say, change it's orientation *and* position may require multiple manipulations using different subsets of the polygon vertices defining that facet. Apparent inversion functions that map user defined offsets proportionally to different vertices would provide more precise control to the manipulation of facet shape, although at the cost of additional degrees of freedom in control that must be specified a priori by the designer.

- **Mapping User Inputs to Motions in X-Y Planes.** Two dimensional user inputs in screen coordinates are mapped into two dimensional offsets in an (x, y) plane in configuration space. The added flexibility of mapping user inputs to offsets in (x, θ), (y, θ), and arbitrarily specified planes would have been useful in modifying contact facet shapes with fewer selection - modification steps.

6.2.3 Some Remaining Issues

Uncertainty and Reliability of Designs

In most of our examples we have assumed that all design parameters, both shape and non-shape, are known exactly. An important concern in any design activity is reliability – how well does a design perform when parameters vary from their nominal values? More generally, how can we model the effects of uncertainty on the functional behavior of artifacts and include the evaluation of these effects in the design process?

Uncertainty is a particularly important consideration for shape parameters because, as we have seen in a number of examples, small variations in critical shape parameters can produce motion constraints that are topologically, and therefore functionally, quite different. For example, in Section 3.1 we noted that by shrinking slightly the width of a tight clearance hole in a peg-in-hole assembly, the corresponding hole on the surface of the CS in configuration space could be made to disappear.[2]

One means of representing uncertainty in shape parameters would be to define motion constraint surfaces where each shape parameter takes on a range of values. For example, if we were to represent each polygon vertex as a small bounded region in (x, y), then the resulting contact facets could be represented as bounded *volumes* in (x, y, θ) configuration space where the surfaces of these volumes correspond to facet equations evaluated at extreme values of the vertex parameters [13]. The

[2] More generally, it is possible to change the genus of the CS surface, i.e. produce holes and embedded free regions, by parametrically modifying the shapes of interacting objects of fixed genus-0. We again feel compelled to note that the sensitivity of the constraint representations to parametric variations is a reflection of the inherent nature of motion constraints, and is **not** a limitation of the chosen representation or an artifact of the implementation.

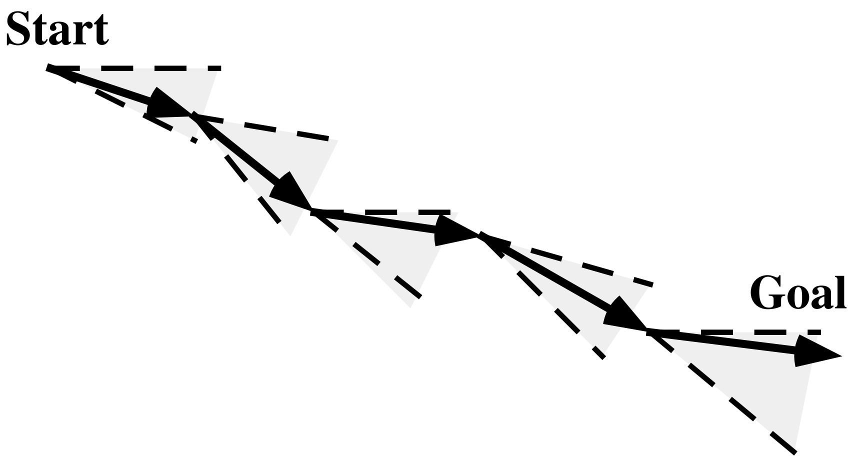


Figure 6.1: A nominal motion path, with depth-one branches at each point where a dynamic parameter variation would result in a different instantaneous motion.

construction and interpretation of such representations would, however, be extremely complex. Another means for representing shape uncertainty would be to extend the present (x, y, θ) representation to include dimensions for parameter variations [23]. This, too, would seem inappropriate for our purposes since it would make the process of design even less tractable.

Another means of representing shape uncertainty, at least indirectly, would be to discretely sample the set of shape parameters in much the same way as was done for the dynamics parameters for compliant assembly in Section 4.5. Specifically, since we have in place the tools necessary to change shape parameters and view the effects of these changes on the motion constraints, we can simply use these tools to evaluate the effects of shape uncertainty directly. In essence, we are using dynamic constraint visualization as a means of visualizing the sensitivity of a design to variations in shape parameters (see Section 4.1.3). Although this places an extra burden on the designer, it nevertheless avoids a considerable degree of added complexity that would result from more direct methods.

In addition to shape parameters, we want to consider the effects of uncertainty on parameters such as the direction of the applied force, the coefficient of friction μ, or the initial position from which a motion is to start. Since these parameters only affect the forward projections, we need to consider ways of extending the forward projection representations to include uncertainty. One simple method, mentioned earlier for compliant assemblies, was to form path *bundles* corresponding to discrete samples of the parameters in question.

Another way of representing uncertainty for a motion path given a set of dynamics parameters represented as nominal values plus ranges on those values would be to compute the exact motion paths using the nominal values as before, but at each step determine if any combination of parameter values in the given ranges would cause the instantaneous motion at that point to vary beyond a specified range. If so, then in addition to the nominal path we could compute the *extreme bifurcations* from the path at that point. The result would be a nominal path, as before, but with a series of branches at those points along the path where some combination of parameters would have significantly affected the forward projection. The path would form a constant depth tree, where each branch represented the beginning of a potential alternate path. Figure 6.1 illustrates such a path, where generally speaking the "hairier" the path, the more sensitive it is to variations in dynamics parameters.[3] Recalling our discussion of constraint features on the CS in Section 4.4.2, we note that motion paths along valleys on the CS are generally quite stable and insensitive to variations in dynamics parameters. Therefore, we would expect branches to occur along portions of a path that cross individual facets.

Yet another means of visualizing the robustness of a portion of a path to variations in dynamics parameters would be to determine the range of parameter values for which the instantaneous motion at a specified point on a path would remain within a given bound. In other words, rather than determining the behavior of a path to specified bounds on the dynamics parameters, we could compute the ranges of parameters that would be guaranteed to keep a local motion within specified bounds on the path. The narrower the range of parameters for a given point, the more sensitive that portion of the path is to parametric variations.

For bounded energy forward projections, the inclusion of uncertainty is somewhat more straightforward since we may either expand or contract the convex forward projection cones appropriately to compensate for ranges of parameter values such as the coefficient of restitution e. With the exception of path bundles for compliant assembly, none of these uncertainty representations for dynamics parameters have been implemented.

Potential for Automated Design

In the previous examples we have seen how the motion constraint representations and design functions may be used by a designer to visualize, analyze and manipulate design parameters to perform design. As noted earlier in Section 4.1.5, the motion constraint representations can be viewed as mathematically precise and computationally accessible data structures. Clearly, then, these representations should be amenable to some level of automated design.

[3] We are neglecting the fact that small errors near the beginning of a path may gradually increase to produce large errors at the goal.

What we have also seen in the previous examples is a high degree of very non-linear coupling between parameter modifications and changes to motion constraints, particularly for the motion constraints found in the vibratory bowl feeder examples. What algorithms might have a reasonable chance of success for such a domain? Two paradigms that come to mind are:

- **Iterative generate and test.** Perhaps the simplest approach to automated design would be the semi-random perturbation of design variables coupled with a binary *go/no-go* filter on the resulting designs.

- **Non-linear optimization.** Another option would be to compute and impose some form of appropriate metric on the motion constraint representations (and by means of apparent-inversion on the underlying design variables) that would be maximized or minimized.

The first approach has the obvious appeal of simplicity. One serendipitous advantage of having to implement the motion constraint representations for use in interactive design is the fact that they may be computed **quickly**. For example, because we can compute the full set of motion constraints for a typical feeder example, including forward projections, on the order of once per second, it is not unreasonable to assume that a random search of the design space might yield viable designs within a reasonable time frame. The resolution at which we randomly sample will be a particularly important consideration since we do not know how tightly grouped potential designs may be. Of course, all of this assumes that the space of designs is relatively small and viable designs are not too sparsely scattered, which in general may not be the case.

The second approach requires us to construct an additional metric that characterizes, preferably in a continuous sense, a good design. For bowl feeders we are primarily concerned with manipulating the *proximity* of motion paths to unsupported regions on the CS. Specifically, given a desired *accept* path, our goal is to both: **(a)** *maximize* the distance between all points on the accept path and all unsupported regions on the CS surface, and **(b)** *minimize* the distance between points on all *reject* paths and support transition boundaries. In case **(b)**, we ideally want the distances to go to zero, i.e. rejected motions should intersect unsupported regions. Using this description, we could envision a cost function metric based on proximity between paths and unsupported regions in the form of a potential energy function. In this scenario, as the design parameters were varied so as to minimize the cost function, the unsupported regions would act as *sources* to repel the *accept* path and at the same time act as *sinks* to attract the remaining *reject* paths.

A design algorithm based on standard linear optimization techniques using such a cost function, although appealing, would possess some major drawbacks as well. Perhaps the most serious of these is the fact that, due to the inherent complexity of the motion constraints and the nonlinear coupling between variations to the constraints and parameters, there would likely be a number of local minima that would not necessarily correspond to satisfactory design solutions. Therefore, a simple hill climbing algorithm would be unlikely to produce many good designs. Furthermore, there is no guarantee on the stability of or convergence toward a design goal of such an algorithm.

One potential solution is suggested by examining the feeder design methodology outlined by the flowchart in Figure 4.19 that was intended for use by a human designer. Specifically, the nested iterative design loops, each working at a lower level of detail on a smaller subset of design parameters, suggests that local optimization of constraints coupled with an occasional reformulation of the higher level problem, or large scale *jump* in design space, could yield better results than local optimization alone. Therefore, an automated optimization algorithm based on *simulated annealing* or similar strategies may provide the best compromise.

We could utilize a number of simplifications and optimizations by taking advantage of the structure of the motion constraint representations. For example, rather than constructing one complex energy function we may be able to formulate a number of simpler energy metrics based on local proximity between selected points on each path and selected points on nearby support transition boundaries that would allow a linear optimization strategy to quickly converge to local minima. These local energy metrics would have to be recomputed for each significant change in the topology of the motion constraint representations, which would occur more or less frequently depending on the current *annealing temperature*. We could also take advantage of the coupling sensitivity information available from the implemented design functions to compute local gradients in the design space to help determine in which direction larger scale parametric changes might be more likely to achieve a desired design. Once again, we cannot guarantee that such an algorithm would converge to a suitable design. Nor can we say anything definite about the rate of convergence of such an optimization strategy.

Finally, what about automated design algorithms for compliant assembly? As in the case of interactive design, the set of constraints that must be manipulated is smaller and less tightly coupled than for bowl feeders. A brute force generate and test design paradigm might therefore be better suited to assembly since we are working over a smaller range of motion constraints. An energy based optimization metric similar to the one described above in which the assembly goal acts as a *sink* would also appear to be a reasonable strategy to investigate. One complicating factor, as mentioned earlier in Section 4.5, is that the addition of non-assembly constraints would seem to be an important consideration since unconstrained automated algorithms that only optimize on the basis of motion constraints would tend to turn everything into a pair nesting cones with the compliant center at the moving part's tip. Labeling certain object features as *fixed* would presumably maintain non-assembly constraints, but leaving these potentially over-conservative constraints "untouchable" by the algorithm could often result in no satisfactory solutions being found.

Many of the above issues, as well as working implementations of automated design strategies along the lines of those described above, remain issues for further research.

6.3 Future Work

Near Term

The limitations of the present implementation of `cspace-shell` suggest some nearer term enhancements that could improve the power and usefulness of the representations and tools. Additional enhancements and extensions might include the following.

- Complete the development and implementation of the energy bounded forward projections necessary to model the jig and APOS feeder application domains. The main challenge here is the fact that implementing these forward projections would require us to extend the current incremental CS topology construction to compute a more complete representation of the global CS topology, similar to the approach used by Brost [13]. Since this computation would almost certainly require more time, we would most likely have to adopt the present approach of computing CS topology only as a post-processing step after an interactive shape modification operation has been completed.

- Implement the superposition of motion constraints generated for planar polygons representing multiple *slices* of three dimensional objects taken at different z-heights.

- Expand upon the quasi-static dynamics model for exact motion integration to include dynamic effects due to inertia. Depending on the operating conditions for the bowl feeder and assembly application domains, more detailed dynamic models may improve the simulation and verification of object motions.

Longer Term

All of the above implementational enhancements, along with most of those listed in the previous section, are well within the scope of the existing representations and algorithms. We might view them as features that we *should* have known to implement, or approaches that we *should* have adopted earlier on in the research.

Other enhancements are sufficiently different or challenging to be classified as significant new research directions based on the current work. Some of these include:

- Implement the design functions necessary to allow direct manipulation of motion paths vs. motion constraints as discussed in the previous section.

- Develop and implement representations for uncertainty and reliability in the design of motion constraints along the lines of those outlined in Section 6.2.3.

- Develop and implement automated and semi-automated design techniques along the lines of those discussed in Section 6.2.3.

- Explore other application domains beyond the four developed in Chapter 2, and extend the representations and tools as necessary. Examples of potentially promising domains include the design of tools and fasteners as illustrated in Chapter 1, electrical connectors and couplings, multiple degree of freedom mechanisms, ...

- Expand and integrate the motion constraint representations with other engineering representations and analysis/design techniques such as: analytic and numerical models of cost, material strength and stiffness, dynamics and vibration, and manufacturing processes such as machining and casting. Examples from the domain of assembly examined in Section 4.5 clearly illustrated the need for such models.

- A considerably more challenging extension than the superposition of multiple planar slices of a 3D object would be to model three dimensional objects directly as polyhedra, and compute the motion constraints for interactions among those polyhedra constrained to move in (x, y, θ). Type A and type B facets would correspond to face-vertex and vertex-face interactions between the moving and stationary polyhedra, respectively, and a new facet type representing edge-edge interactions (type C) would also be required (see Lozano-Pérez [49]).

- Beyond polyhedra, a significant challenge would be to generate and render motion constraints produced by more general shape interactions, such as objects modeled using bi-cubic patches.

- Finally, we should by no means by committed to the motion constraint representation or interactive design environment if another representation appears to hold more promise in simplifying the tasks of analysis and design. For example, it may be possible to develop hybrid functional metaphor/motion constraint representations that could ease the designer's burden of learning new and unfamiliar representations.

Energy Bounded Forward Projections

Appendix A

A.1 Forward Projections for Dropped Objects

This section derives the models used to compute the forward projection of a dropped particle for single and double impacts with oriented surfaces in a gravity field.

Single Bounce

The profile for the maximum height reached after the initial bounce of a particle (h_{max1} in Figure 2.7) as a function of the impact surface orientation φ_1 is derived as

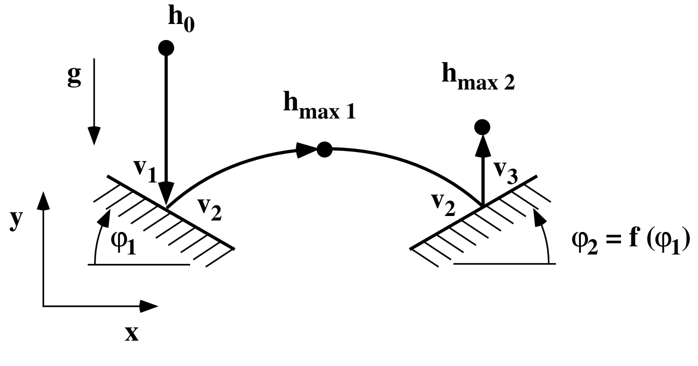


Figure A.1: Model of single and double impacts for a dropped particle.

follows.

The pre-impact velocity v_0 of the particle is given by conservation of energy, assuming zero initial velocity, as:

$$\frac{1}{2}mv_0^2 = mgh_0,$$

so that

$$v_0 = \sqrt{2gh_0}.$$

As a conservative approximation, we assume the surface to be frictionless. We then compute the post-impact velocity components using the coefficient of restitution e and equation 2.1:

$$\begin{aligned} v_{1x} &= v_0(1+e)\sin\varphi_1\cos\varphi_1 \\ v_{1y} &= v_0(e\cos^2\varphi_1 - \sin^2\varphi_1). \end{aligned}$$

The post-impact motion of the particle is parabolic:

$$\begin{aligned} x(t) &= x_0 + v_{1x}t \\ y(t) &= y_0 + v_{1y}t - \frac{g}{2}t^2. \end{aligned}$$

Finally, at the point of maximum height we have:

$$\begin{aligned} t_{y_{max}} &= \frac{v_{1y}}{g} \\ x_{y_{max}} &= x_0 + v_{1x}t_{y_{max}} = x_0 + \frac{v_0^2}{g}\Big((1+e)\sin\varphi_1\cos\varphi_1(e\cos^2\varphi_1 - \sin^2\varphi_1)\Big) \\ y_{max} &= y_0 + \frac{v_{1y}^2}{2g} = y_0 + \frac{v_0^2}{2g}\left(e\cos^2\varphi_1 - \sin^2\varphi_1\right)^2. \end{aligned}$$

Double Bounce

The profile for the maximum height reached after the second bounce of a particle (h_{max2} in Figure 2.7) as a function of the impact surface orientation φ_1 is derived as follows.

Assuming the height of the second point of impact is the same as the first:[1]

$$v_{2x} = v_{1x} = v_0(1+e)\sin\varphi_1\cos\varphi_1$$

[1]The equations for the case where the second impact is at a different height from the first are considerably more complex, but the solutions indicate that the forward projection derived from the equal height analysis given here contains the other paths.

$$v_{2y} \quad = \quad -v_{1y} = -v_0(e\cos^2\varphi_1 - \sin^2\varphi_1).$$

and

$$|\vec{v_2}| = |\vec{v_1}| = v_0\sqrt{\sin^2\varphi_1 + e^2\cos^2\varphi_1}.$$

We require the velocity after the second impact to be vertical so that

$$v_{3x} = 0.$$

From this constraint and from the symmetry of the problem we have the following relationship between φ_1 and φ_2:

$$\varphi_1 + \tan^{-1}\left(\frac{1}{e}\tan\varphi_1\right) = \tan^{-1}\left(\frac{1}{e}\tan\varphi_2\right) + \varphi_2$$

We may solve the resulting expression numerically to achieve:

$$\varphi_2 = f(\varphi_1)$$

We may now perform the same pre and post-impact analysis as was don for the first bounce. The result is

$$v_{3y} = |\vec{v_3}| = v_2\left(\sin\psi\sin\varphi_2 + e\cos\psi\cos\varphi_2\right)$$

where

$$\psi = \tan^{-1}(e\tan\varphi_2)$$

After rearranging we have

$$v_3 = v_2\frac{e\cos\psi}{\cos\varphi_2}$$

where

$$|\vec{v_2}| = \sqrt{2gh_0\sin^2\varphi_1 + e^2\cos^2\varphi_1}$$

Finally, from conservation of energy we have

$$h_{max2} = \frac{v_3^2}{2g}$$

where, from above, $v_3 = g(\varphi_1)$.

A.2 Non-Conservative Bouncing

In this section we estimate the maximum height attainable by a particle starting from rest on a table with coefficient of restitution e, and vibrating with frequency ω_t and amplitude A_0. We compute the maximum height by considering the most extreme of impact conditions. Specifically, we assume that after falling from the maximum height, the energy lost by the particle during impact with the table exactly balances the maximum energy imparted to the particle by the table (i.e. when the table is moving at its maximum upward velocity).

From equation 2.1 we have:

$$v_{table} - v_2 = -e(v_1 + v_{table})$$

where v_1 and v_2 are the pre and post-impact velocities of the particle, respectively.

From an energy balance on the particle under the above assumptions we have:

$$v_2 = v_1$$

So that the maximum post-impact velocity of the particle is

$$v_{particle_{max}} = \frac{(1+e)}{(1-e)} v_{table}.$$

The maximum velocity of the table is

$$v_{table_{max}} = A_0 \omega.$$

From conservation of energy after the particle leaves the table:

$$H_{max} = \frac{v^2_{particle_{max}}}{2g}$$

so that finally we have

$$H_{max} = \frac{(A_0\omega)^2}{2g} \frac{(1+e)^2}{(1-e)^2}.$$

Facet Curvature

Appendix B

In this appendix we derive the maximum motion integration stepsize based on a specified upper error bound E_{max}.

We begin with a derivation of the general curvature κ_n along a 3D surface from Faux & Pratt [30]. The surface is defined as a vector quantity $\vec{r}$ where

$$\vec{r} = x\underline{i} + y\underline{j} + z\underline{k}$$

A curve on a (u, v) surface is given by $u = u(t)$ and $v = v(t)$ where t is a parameter along the curve. The surface normal at a given point on the surface is given in terms of the parameters (u, v) by

$$\vec{n} = \frac{\left(\frac{\partial \vec{r}}{\partial u} \times \frac{\partial \vec{r}}{\partial v}\right)}{\left|\frac{\partial \vec{r}}{\partial u} \times \frac{\partial \vec{r}}{\partial v}\right|}.$$

The *curvature* κ_n of the surface along the curve is given by

$$\kappa_n = \frac{\dot{\underline{u}}^T \overline{D} \dot{\underline{u}}}{\dot{\underline{u}}^T \overline{G} \dot{\underline{u}}}$$

where

$$\underline{u} = [u(t), v(t)]^T,$$

and $\overline{D}$ is the *second fundamental matrix* given by

$$\overline{D} = \begin{bmatrix} \vec{n} \cdot \frac{\partial^2 \vec{r}}{\partial u^2} & \vec{n} \cdot \frac{\partial^2 \vec{r}}{\partial u \partial v} \\ \vec{n} \cdot \frac{\partial^2 \vec{r}}{\partial v \partial u} & \vec{n} \cdot \frac{\partial^2 \vec{r}}{\partial v^2} \end{bmatrix}$$

and $\overline{G}$ is the *first fundamental matrix* given by

$$\overline{G} = \begin{bmatrix} \frac{\partial \vec{r}}{\partial u} \cdot \frac{\partial \vec{r}}{\partial u} & \frac{\partial \vec{r}}{\partial u} \cdot \frac{\partial \vec{r}}{\partial v} \\ \frac{\partial \vec{r}}{\partial v} \cdot \frac{\partial \vec{r}}{\partial u} & \frac{\partial \vec{r}}{\partial v} \cdot \frac{\partial \vec{r}}{\partial v} \end{bmatrix}$$

Carrying out the matrix multiplication and simplifying we have

$$\kappa_n = \frac{d_{11}\dot{u}^2 + 2d_{12}\dot{u}\dot{v} + d_{22}\dot{v}^2}{g_{11}\dot{u}^2 + 2g_{12}\dot{u}\dot{v} + g_{22}\dot{v}^2}$$

where $\dot{u} = \frac{du}{dt}$, $\dot{v} = \frac{dv}{dt}$, and d_{ij} and g_{ij} are elements of $\overline{D}$ and $\overline{G}$ respectively.

For Type A and Type B contact facets, the derivation of the curvature κ_n is given as follows. For facets parameterized by (p, θ), a curve on the (u, v) surface is given by $u = p(t)$ and $v = \theta(t)$, where as before t is a parameter along the curve.

The elements d_{ij} and g_{ij} of $\overline{D}$ and $\overline{G}$ are:

$$\begin{aligned} d_{11} &= n_x \frac{\partial^2 x}{\partial p^2} + n_y \frac{\partial^2 y}{\partial p^2} + n_z \frac{\partial^2 z}{\partial p^2} \\ d_{12} = d_{21} &= n_x \frac{\partial^2 x}{\partial p \partial \theta} + n_y \frac{\partial^2 y}{\partial p \partial \theta} + n_z \frac{\partial^2 z}{\partial p \partial \theta} \\ d_{22} &= n_x \frac{\partial^2 x}{\partial \theta^2} + n_y \frac{\partial^2 y}{\partial \theta^2} + n_z \frac{\partial^2 z}{\partial \theta^2} \end{aligned}$$

and

$$\begin{aligned} g_{11} &= \left(\frac{\partial x}{\partial p}\right)^2 + \left(\frac{\partial y}{\partial p}\right)^2 + \left(\frac{\partial z}{\partial p}\right)^2 \\ g_{12} = g_{21} &= \frac{\partial x}{\partial p}\frac{\partial x}{\partial \theta} + \frac{\partial y}{\partial p}\frac{\partial y}{\partial \theta} + \frac{\partial z}{\partial p}\frac{\partial z}{\partial \theta} \\ g_{22} &= \left(\frac{\partial x}{\partial \theta}\right)^2 + \left(\frac{\partial y}{\partial \theta}\right)^2 + \left(\frac{\partial z}{\partial \theta}\right)^2. \end{aligned}$$

We recall from Section 5.3.1 that the elements of $\vec{r}$ for the contact facets are of the form:

$$\begin{aligned} x &= f_x(p, \theta) \\ y &= f_y(p, \theta) \\ z &= a\theta \end{aligned}$$

Recalling the *move* and *turn* convention of Section 5.3.2, we use the following notation:

$$\begin{aligned}
mx &= move_x \\
my &= move_y \\
tx &= turn_x \\
ty &= turn_y \\
sx &= start_x \\
sy &= start_y
\end{aligned}$$

where *start* is the (x, y) start point of a move. Using this notation, we have for a Type A Facet:

$$\begin{aligned}
x &= tx + sx\cos\theta - sy\sin\theta + p(mx\cos\theta - my\sin\theta) \\
y &= ty + sx\sin\theta + sy\cos\theta + p(mx\sin\theta + my\cos\theta) \\
z &= a\theta
\end{aligned}$$

and for a Type B Facet:

$$\begin{aligned}
x &= sx + tx\cos\theta - ty\sin\theta + pmx \\
y &= sy + tx\sin\theta + ty\cos\theta + pmy \\
z &= a\theta.
\end{aligned}$$

Using these equations, we can now derive the terms necessary to compute the elements d_{ij} and g_{ij} given above.

For a Type A Facet we have:

$$\begin{aligned}
\frac{\partial x}{\partial p} &= mx\cos\theta - my\sin\theta \\
\frac{\partial^2 x}{\partial p^2} &= 0 \\
\frac{\partial x}{\partial \theta} &= -sx\sin\theta - sy\cos\theta + p(-mx\sin\theta - my\cos\theta) \\
\frac{\partial^2 x}{\partial^2 \theta} &= -sx\cos\theta + sy\sin\theta + p(-mx\cos\theta + my\sin\theta) \\
\frac{\partial^2 x}{\partial p \partial \theta} &= -mx\sin\theta - my\cos\theta \\
\frac{\partial y}{\partial p} &= mx\sin\theta + my\cos\theta
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 y}{\partial p^2} &= 0 \\
\frac{\partial y}{\partial \theta} &= sx\cos\theta - sy\sin\theta + p(mx\cos\theta - my\sin\theta) \\
\frac{\partial^2 y}{\partial^2 \theta} &= -sx\sin\theta - sy\cos\theta + p(-mx\sin\theta - my\cos\theta) \\
\frac{\partial^2 y}{\partial p \partial \theta} &= mx\cos\theta - my\sin\theta) \\
\frac{\partial z}{\partial p} &= 0 \\
\frac{\partial^2 z}{\partial p^2} &= 0 \\
\frac{\partial z}{\partial \theta} &= a \\
\frac{\partial^2 z}{\partial^2 \theta} &= 0 \\
\frac{\partial^2 z}{\partial p \partial \theta} &= 0.
\end{aligned}$$

For a Type B Facet we have:

$$\begin{aligned}
\frac{\partial x}{\partial p} &= mx \\
\frac{\partial^2 x}{\partial p^2} &= 0 \\
\frac{\partial x}{\partial \theta} &= -tx\sin\theta - ty\cos\theta \\
\frac{\partial^2 x}{\partial^2 \theta} &= -tx\cos\theta + ty\sin\theta \\
\frac{\partial^2 x}{\partial p \partial \theta} &= 0
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y}{\partial p} &= my \\
\frac{\partial^2 y}{\partial p^2} &= 0 \\
\frac{\partial y}{\partial \theta} &= tx\cos\theta - ty\sin\theta \\
\frac{\partial^2 y}{\partial^2 \theta} &= -tx\sin\theta - ty\cos\theta
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 y}{\partial p \partial \theta} &= 0 \\
\frac{\partial z}{\partial p} &= 0 \\
\frac{\partial^2 z}{\partial p^2} &= 0 \\
\frac{\partial z}{\partial \theta} &= a \\
\frac{\partial^2 z}{\partial^2 \theta} &= 0 \\
\frac{\partial^2 z}{\partial p \partial \theta} &= 0.
\end{aligned}$$

In Section 5.4.2 we developed the expressions needed to compute the instantaneous direction of motion on a facet surface, in terms of Δp and $\Delta\theta$, from the net force components at the point of contact. Using Equations 5.45 and 5.46 we can approximate $\dot{u}$ and $\dot{v}$ as:

$$\begin{aligned}
\dot{u} &= \frac{dp}{dt} \approx \frac{\Delta P}{\Delta T} = \frac{\Delta P}{\sqrt{\Delta P^2 + \Delta\theta^2}} \\
\dot{v} &= \frac{d\theta}{dt} \approx \frac{\Delta\theta}{\Delta T} = \frac{\Delta\theta}{\sqrt{\Delta P^2 + \Delta\theta^2}}.
\end{aligned}$$

Using this approximation for $\dot{u}$ and $\dot{v}$, and the elements d_{ij} and g_{ij} computed from the above expressions for the appropriate facet type, we compute the facet curvature κ_n along the specified $(\Delta p, \Delta\theta)$ direction of motion across the facet. From the facet curvature, we compute the equivalent *radius of curvature* R as

$$R = \frac{1}{\kappa_n}.$$

Figure B.1 shows the relationship between a specified maximum error E_{max}, the radius of curvature R along the motion path, and the maximum stepsize ΔT_{max}. From the figure we see that

$$\Delta T_{max} = R \sin\phi_{max}$$

and

$$E_{max} = R(1 - \cos\phi).$$

With some simple trigonometry and rearranging we have

$$\cos\phi_{max} = 1 - \frac{E}{R}$$

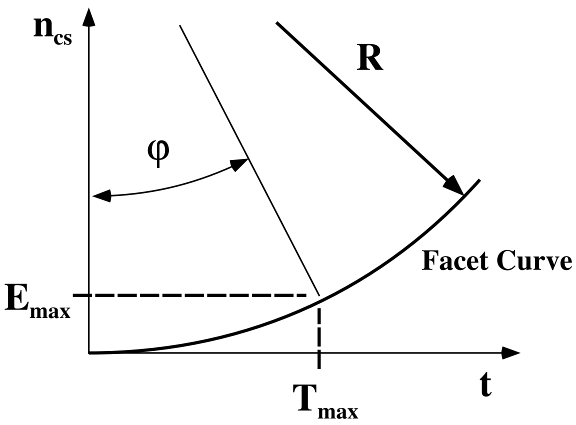


Figure B.1: Determining the maximum integration stepsize on a curved facet.

and

$$\begin{aligned}
\sin\phi_{max} &= \sqrt{1-\cos^2\phi_{max}} \\
&= \sqrt{1-\left(1-\frac{E}{R}\right)^2} \\
&= \frac{1}{R}\sqrt{2ER-E^2}.
\end{aligned}$$

Finally we have

$$\begin{aligned}
\Delta T_{max} &= R\sin\phi_{max} = \sqrt{2E_{max}R-E_{max}^2} \\
&= \sqrt{\frac{2E_{max}}{\kappa_n}-E_{max}^2}
\end{aligned}$$

Given ΔT_{max}, we can compute the maximum stepsize in terms of the facet parameters (p,θ) using Equations 5.48 and 5.49 in Section 5.4.2, which we repeat here for reference

$$\begin{aligned}
\Delta P_{max} &= \left(\frac{\Delta p}{\sqrt{\Delta p^2+\Delta\theta^2}}\right)\Delta T_{max} \\
\Delta\theta_{max} &= \left(\frac{\Delta\theta}{\sqrt{\Delta p^2+\Delta\theta^2}}\right)\Delta T_{max}.
\end{aligned}$$

Primary Data Structures

Appendix C

For Representing Objects:

`PART` An array of (x, y) polygon vertices and reference point **cg** describing a planar polygon.

```
typedef struct {
  PPOINT  points[MAX_PART_POINTS];  /* Array of points           */
  int     point_count;              /* Number of vertex points   */
  int     type;                     /* Type of part (A or B)     */
  PPOINT  cg;                       /* Centroid of the part (A)  */
  char    *name;                    /* Name of part              */
} PART, *PART_PTR;
```

`TRACE` An array of MOVEs and TURNs describing a planar polygon.

```
typedef struct {
  MOVE     moves[MAX_PART_POINTS];   /* An array of moves */
  int      move_count;               /* Number of moves   */
  TURN     turns[MAX_PART_POINTS];   /* An array of turns */
  int      turn_count;               /* Number of turns   */
} TRACE, *TRACE_PTR;
```

`MOVE` A pair of (x, y) vertices forming a polygon edge and the length and angle of the edge.

```
typedef struct {
  int     type;       /* Type of move (from partA or partB) */
  int     direction;  /* FORWARD_MOVE or BACKWARD_MOVE */
  PPOINT  startpt;    /* Part point relative to c.g. (Includes original partpt index) */
  PPOINT  endpt;      /* Part point relative to c.g. (Includes original partpt index) */
  double  angle;      /* Original orientation of move in radians */
  double  length2;    /* Length (squared) of move */
  PPOINT  component;  /* X,Y components of segment (start to end)   */
} MOVE, *MOVE_PTR;
```

`TURN` An (x, y) vertex point, with angles for the entering edge and exiting polygon edge, and a LEFT_TURN or RIGHT_TURN corresponding to a convex or concave vertex.

```
typedef struct {
  int     type;           /* Type of turn (from partA or partB) */
  int     direction;      /* LEFT_TURN or RIGHT_TURN */
  PPOINT  turnpt;         /* Part point relative to c.g. (Includes original partpt index) */
  double  start_angle;    /* Original orientation of move entering turn in radians */
  double  end_angle;      /* Original orientation of move leaving turn in radians */
} TURN, *TURN_PTR;
```

For Representing Motion Constraints:

`CS` An array of $2NM$ type A and type B contact facets.

```
typedef struct {
  FACET  facetsA[MAX_MOVES][MAX_TURNS];  /* FACET array (2D) indexed by MOVE and TURN */
  int    facetsA_moves;                  /* Number of facetsA moves (partA moves) */
  int    facetsA_turns;                  /* Number of facetsA turns (partB turns) */
  FACET  facetsB[MAX_MOVES][MAX_TURNS];  /* FACET array (2D) indexed by MOVE and TURN */
  int    facetsB_moves;                  /* Number of facetsB moves (partB moves) */
  int    facetsB_turns;                  /* Number of facetsB turns (partA turns) */
} CS, *CS_PTR;
```

`FACET` Contains information on facet type and size for rendering a contact facet, arrays of `REL_FACET`s containing adjacent and intersecting facets for computing motion paths, and a support map to record the support status of points on the facet surface.

```
typedef struct {
  int           type;                 /* Type of FACET (TYPE_A or TYPE_B) */
  int           direction;            /* FORWARD_MOVE or BACKWARD_MOVE facet */
  int           move_index;           /* Index to part MOVE forming this facet */
  MOVE_PTR      move;                 /* Pointer to MOVE forming this facet (for speed) */
  int           turn_index;           /* Index to part TURN forming this facet */
  TURN_PTR      turn;                 /* Pointer to TURN forming this facet (for speed) */
  double        min_theta;            /* Minimum theta for which this facet is valid */
  double        max_theta;            /* Maximum theta for which this facet is valid */
  int           adjacents_posted;     /* Flag indicating if adjacents have been posted */
  REL_FACET_PTR  adjacents;           /* BEGINNING of the array of adjacents */
  int           adjacents_count;      /* Number of adjacent facets */
  int           intersects_posted;    /* Flag indicating if intersects have been posted */
  REL_FACET_PTR  intersects;          /* BEGINNING of the array of intersects */
  int           intersects_count;     /* Number of intersecting facets */
  int           support_status;       /* (SUPPORTED, UNSUPPORTED, TEST_SUPPORT) */
  SUPPORT_PTR   support_map;          /* Beginning of array of support transitions */
  int           support_map_count;    /* Number of support_map elements */
  double        support_delta_theta;  /* Theta step at which support map is computed */
  double        support_delta_p;      /* Nominal p step at which support map is computed */
} FACET, *FACET_PTR;
```

`SUPPORT` Contains an array of support transitions for a given fixed-θ slice of a facet.

```
typedef struct {
  double   theta;                  /* Value of facet theta parameter at which slice is taken */
  int      p0_status;              /* Support status at facet parameter (p = 0.0) */
  int      p1_status;              /* Support status at facet parameter (p = 1.0) */
  SUPPORT_TRANS_PTR  support_trans; /* Beginning of the support_trans array */
  int      support_trans_count;     /* Number of elements in support_trans array */
} SUPPORT, *SUPPORT_PTR;
```

`SUPPORT_TRANS` A flag determining whether the moving polygon *gains* or *loses* support at a given p position moving in the $+p$ direction, together with the value of $p = (0, 1)$.

```
typedef struct {
  int     transition_type;  /* Type of support transition (GAIN_SUPPORT or LOSE_SUPPORT) */
  double  p_facet;          /* Value of facet p parameter at which transition is marked */
  int     track_segment_id; /* Index to poly segment of the track */
  double  p_track;          /* Value of track segment p parameter at which trans. marked */
} SUPPORT_TRANS, *SUPPORT_TRANS_PTR;
```

For Representing and Computing Object Motions:

`PATH` An array of `CONTACT_STATE`s, with a termination flag and an initial condition probability $(0 \rightarrow 1)$

```
typedef struct {
  CONTACT_STATE_PTR  states;           /* BEGINNING of the array of CONTACT_STATEs */
  int                state_counter;    /* No. of states in PATH */
  int                termination_flag; /* Flag indicating termination status of path */
  double             probability;      /* Probability of part entering this path */
} PATH, *PATH_PTR;
```

`CONTACT_STATE` An array of `REL_FACET`s containing those facets with which the current set of positions along the path are in contact, an array of positions *on* the facets, and an array of positions slightly *above* the facets for displaying the path.

```
typedef struct {
  int         contact_type;              /* Type of contact (see defines) */
  REL_FACET   contacts[MAX_CONTACTS];    /* Array of facets forming contact */
  int         contacts_count;            /* No. of contacting facets */
  REL_FACET_PTR  non_contacts;  /* BEGINNING of array of non-contacting proximal facets */
  int         non_contacts_count;        /* No. of non-contacting facets (and proximals) */
  POS_PTR     positions;        /* BEGINNING of positions array of trajectory points */
  int         positions_count;           /* No. of trajectory points */
  POS_PTR     display_pos;      /* BEGINNING of positions array for displayed points */
} CONTACT_STATE, *CONTACT_STATE_PTR;
```

`FACET_FORCE_STATE` A record of the instantaneous net force state on a single facet at a point along the path, including the position, normal and tangential vectors, and vectors bounding the configuration space friction cone.

```
typedef struct {
  FACET_PTR     facet;       /* Pointer to a FACET */
  POS_PTR       pos;         /* Pointer to POS on facet */
  double        p;           /* Param. P corresponding to pos on FACET */
  double        n_co[3];     /* CS normal at point */
  double        m_p[3];      /* CS m_p at point */
  double        m_theta[3];  /* CS m_theta at point */
  double        n_fc[3];     /* Normal to CS friction cone plane */
  double        f_rhs[3];    /* RHS bounding ray of CS friction cone */
  double        f_lhs[3];    /* LHS bounding ray of CS friction cone */
  double        f_net[3];    /* Net force on contact */
} FACET_FORCE_STATE, *FACET_FORCE_STATE_PTR;
```

`PROXIMAL` The signed distance of a point from the surface of a non-contacting facet measured in a constant-θ plane.

```
typedef struct {
  int     proximal_state;  /* State of facet adjacency or point_to_line proximity */
  double  distance;        /* Signed distance from point_to_line */
} PROXIMAL, *PROXIMAL_PTR;
```

`MOTION_PARAMS` A record of the external force applied to the reference point, the path integration step size, and parameters ρ and μ.

```
typedef struct {
  double    f_ext[3];     /* Externally applied force */
  double    max_step;     /* Maximum step size for integration */
  double    rho;          /* Radius of gyration, scales theta to X-Y */
  double    mu;           /* Coefficient of friction */
} MOTION_PARAMS, *MOTION_PARAMS_PTR;
```

Bibliography

[1] **H. Asada, A. B. By**, "Kinematics of Workpart Fixturing", *IEEE International Conference on Robotics and Automation*, March 1985, pp. 337–345.

[2] **I. I. Artobolevsky**, Mechanisms in Modern Engineering Design, MIR Publishers, Moscow, 1976.

[3] **F. Avinaim, J. D. Boissonnat, B. Faverjon**, "A Practical Exact Motion Planning Algorithm for Polygonal Objects Amidst Polygonal Obstacles", *IEEE International Conference on Robotics and Automation*, April 1988, pp. 1656–1661.

[4] **C. Bajaj, M. Kim**, "Generation of Configuration Space Obstacles: The Case of Moving Algebraic Curves", *International Conference on Robotics and Automation*, Raleigh, NC, April 1987, pp. 979–984.

[5] **J. J. Bausch, K. Youcef-Toumi**, "Kinematic Methods for Automated Fixture Reconfiguration Planning", *International Conference on Robotics and Automation*, May 1990, pp. 1396–1401.

[6] **R. M. C. Bodduluri, J. M. McCarthy**, "Design of Spatial Mechanisms Using Constraint Manifolds", Department of Mechanical Engineering, U. C. Irvine, February 1991.

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

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

[9] **D. A. Bourne, D. Navinchandra, R. Ramaswamy**, "Relating Tolerances and Kinematic Behavior", The Robotics Institute, Carnegie Mellon University, Technical Report CMU-RI-TR-89-10, April 1989.

[10] **W. E. Boyes, Ed.**, Jigs and Fixtures, 2nd. Edition, Society of Manufacturing Engineers, Dearborn, MI, 1982.

[11] **M. S. Branicky, W. S. Newman**, "Rapid Computation of Configuration Space Obstacles", *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 1990, pp. 304–310.

[12] **R. C. Brost**, "Dynamic Analysis of Planar Manipulation Tasks", *IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2247–2254.

[13] **R. C. Brost**, "Analysis and Planning of Planar Manipulation Tasks", PhD Thesis, CMU-CS-91-149, Carnegie Mellon University, January 1991.

[14] **S. J. Buckley**, "Planning and Teaching Compliant Motion Strategies", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 936, January 1987.

[15] **M. E. Caine, T. Lozano-Pérez, W. P. Seering**, "Assembly Strategies for Chamferless Parts", *IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, May 1989, pp. 472-477.

[16] **M. Caine**, "The Effect of Part Shape on Planar Insertion Operations", *Proceedings of the Second ASME Conference on Flexible Assembly Systems*, Chicago, IL, Sept. 1990, pp. 133–138.

[17] **M. E. Caine**, "The Design of Shape from Motion Constraints", PhD Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, January 1993.

[18] **J. F. Canny**, The Complexity of Robot Motion Planning, MIT Press, Cambridge, MA, 1988.

[19] **A. Christian**, *Personal Communication*, November, 1990.

[20] **M. S. Darlow, D. M. Avidar, M. W. Steiner**, "A Procedure for Comparative Evaluation of Design for Assembly Methodologies", *ASME Advances in Design Automation-1987*, DE-Vol. 10-1, 1987, pp. 301–306.

[21] **T. L. De Fazio, et. al.**, "A Prototype for Feature-Based Design for Assembly – Revision 1", CSDL-P-2917, The Charles Stark Draper Laboratory, Cambridge, MA, 1990.

[22] **B. R. Donald**, "Motion Planning with Six Degrees of Freedom", Artificial Intelligence Laboratory, TR-791, Massachusetts Institute of Technology, June 1984.

[23] **B. R. Donald**, "Error Detection and Recovery for Robot Motion Planning with Uncertainty", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 982, July 1987.

[24] **B. Donald, D. Pai**, "On The Motion of Compliantly-Connected Rigid Bodies in Contact, Part II: A System for Analyzing Designs for Assembly", *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 1990, pp. 1756-1762.

[25] **The Charles Stark Draper Laboratory**, *Third Annual Seminar on Advanced Assembly Automation*, Nov. 1982, pp. 8(c).1-8(c).14.

[26] **M. A. Erdmann**, "On Motion Planning with Uncertainty", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 810, May 1984.

[27] **M. A. Erdmann, M. T. Mason**, "An Exploration of Sensorless Manipulation", *IEEE Journal of Robotics and Automation*, Vol. 4, No. 4, 1988, pp. 369–379.

[28] **M. A. Erdmann**, "On Probabalistic Strategies for Robot Tasks", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 1155, March 1990.

[29] **B. Faltings**, "Qualitative Place Vocabularies for Mechanisms in Configuration Space", PhD Thesis, Report No. UIUCDCS-R-87-1360, University of Illinois, July 1987.

[30] **I. D. Faux, M. J. Pratt**, Computational Geometry for Design and Manufacture, John Wiley & Sons, New York, 1979.

[31] **J. Foley, et. al.** Computer Graphics – Principles and Practice, 2nd. Edition, Addison-Wesley, Reading, MA, 1990.

[32] **T. Fujimori**, "Development of Flexible Assembly System Smart", *Proc. Int. Symposium on Industrial Robotics*, October 1990, pp. 75–82.

[33] **B. J. Gilmore, D. A. Streit**, "A Rule-Based Algorithm to Predict the Dynamic Behavior of Mechanical Part Orienting Operations", *IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1988, pp. 1254–1259.

[34] **K. Y. Goldberg**, "Stochastic Plans for Robotic Manipulation", PhD Thesis, CMU-CS-90-161, Carnegie Mellon University, August 1990.

[35] **S. J. Gordon**, "Automated Assembly Using Feature Localization", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 932, December 1986.

[36] **J. Guckenheimer, P. Holmes**, Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields, Third Printing, Springer-Verlag, New York, 1983.

[37] **L. Guibas, L. Ramshaw, J. Stolfi**, "A Kinetic Framework for Computational Geometry", *Proceedings of the 24th Annual IEEE Conference on the Foundations of Computer Science*, Tuscon, AZ, 1983, pp. 100–111.

[38] **R. Gupta, M. J. Jakiela**, "Qualitative Simulation of Kinematic Pairs via Small-Scale Interference Detection", *ASME Fourth International Conference on Design Theory and Methodology*, Scottsdale, AZ, September 1992.

[39] **D. A. Hoeltzel, W. H. Chieng**, "Pattern Matching Synthesis as an Automated Approach to Mechanism Design", *Proceedings of the 1989 ASME Design Technical Conferences – 15th Design Automation Conference*, DE-Vol. 19-1, Montreal, Quebec, Canada, Sept. 1989, pp. 243–250.

[40] **S. Goyal**, "Planar Sliding of a Rigid Body with Dry Friction: Limit Surfaces and Dynamics of Motion", PhD Thesis, Dept. of Mechanical Engineering, Cornell University, January 1989.

[41] **Inoue, Hirochika**, "Force Feedback in Precise Assembly Tasks", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, AI Memo-308, Aug. 1974 (Reprinted in Winston, P. H., and Brown, R. H., eds., *Artificial Intelligence: An MIT Perspective*, MIT Press, 1979).

[42] **M. J. Jakiela, P. Y. Papalambros**, "Design and Implementation of an Intelligent CAD System", *ASME Advances in Design Automation-1987*, DE-Vol. 10-1, 1987, pp. 339-345.

[43] **L. Joskowicz, S. Addanki**, "Innovative Shape Design: A Configuration Space Approach",Courant Institute of Mathematical Sciences, New York University, Technical Report 356, March 1988.

[44] **L. Joskowcz**, "Reasoning about Shape and Kinematic Function in Mechanical Devices", PhD Thesis, Technical Report No. 402, New York University, September 1988.

[45] **L. Joskowicz, E. Sachs**, "Unifying Kinematics and Dynamics for the Automatic Analysis of Machines", IBM Technical Report RC 15573, March 1990.

[46] **M. Juodate, A. Saito**, *Personal Communication*, August, 1990.

[47] **G. A. Kramer**, Solving Geometric Constraint Systems: A Case Study in Kinematics, MIT Press, Cambridge, MA, 1992.

[48] **J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg**, "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware", Proceedings of SIGGRAPH '90, Dallas, TX, August 1990.

[49] **T. Lozano-Pérez**, "Spatial Planning: A Configuration Space Approach", *IEEE Transactions on Computers*, Vol. C-32, No. 2, Feb. 1983, pp. 108–120.

[50] **T. Lozano-Pérez, M. T. Mason, R. H. Taylor**, "Automatic Synthesis of Fine-Motion Strategies for Robots", *Proceedings, International Symposium of Robotics Research*, Bretton Woods, NH, MIT Press, Sept. 1984.

[51] **T. Lozano-Pérez**, "Motion Planning and the Design of Orienting Devices for Vibratory Parts Feeders", Unpublished Research Memo, MIT Artificial Intelligence Laboratory, January 1986.

[52] **T. Lozano-Pérez, P. A. O'Donnell**, "Parallel Robot Motion Planning", *IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991, pp. 1000–1007.

[53] **T. Lozano-Pérez, et. al.**, Handey – A Robot Task Planner, MIT Press, Cambridge, MA, 1992.

[54] **M. T. Mason**, "Compliance and Force Control for Computer Controlled Manipulators", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 6, June 1981 (Reprinted in Brady, M. et al., eds., *Robot Motion*, MIT Press, 1983).

[55] **M. T. Mason**, "Manipulator Grasping and Pushing Operations", Artificial Intelligence Laboratory, TR-690, Massachusetts Institute of Technology, June 1982.

[56] **J. M. McCarthy**, Introduction to Theoretical Kinematics, The MIT Press, Cambridge, MA, 1990.

[57] **F. W. Miller**, "Design for Assembly: Ford's Better Idea to Improve Products", Manufacturing Systems, March 1988.

[58] **P. H. Moncevicz**, "Orientation and Insertion of Randomly Presented Parts Using Vibratory Agitation", S.M. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, June 1991.

[59] **B. K. Natarajan**, "Some Paradigms for the Automated Design of Parts Feeders", *International Journal of Robotics Research*, Vol. 8, No. 6, December 1989, pp. 98–109.

[60] **J. L. Nevins, D. E. Whitney, et. al.**, Concurrent Design of Products & Processes, McGraw-Hill, New York, 1989.

[61] **N. M. Patrikalakis (Ed.)**, Scientific Visualization of Physical Phenomena, Springer-Verlag, New York, 1991.

[62] **B. Paul**, Kinematics and Dynamics of Planar Machinery, Prentice-Hall, NJ, 1979.

[63] **M. A. Peshkin**, "Planning Robotic Manipulation Strategies for Sliding Objects", PhD Thesis, Department of Physics, The Robotics Institute, Carnegie Mellon University, 1986.

[64] **F. P. Preparata, M. I. Shamos**, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.

[65] **A. A. G. Requicha, J. R. Rossinac**, "Solid Modeling and Beyond", *IEEE Computer Graphics and Applications*, Vol. 12, No. 5, September 1992, pp. 31–44.

[66] **F. Reuleaux**, The Kinematics of Machinery: Outline of a Theory of Machines, 1876 (Reprinted by Dover Publications, Inc., 1963).

[67] **A. J. Scarr, D. H. Jackson, R. S. McMaster**, "Product Design for Robotic and Automated Assembly", *IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986, pp. 796-802.

[68] **J. M. Schimmels, M. A. Peshkin**, "Admittance Matrix Design for Force-Guided Assembly", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 2, April 1992, pp. 213–227.

[69] **J. E. Shigley, J. J. Uicker, Jr.**, Theory of Machines and Mechanisms, McGraw-Hill, New York, 1980.

[70] **M. Shirai, A. Saito**, "Parts Supply in Sony's General-Purpose Assembly System "SMART"", *Jpn. J. Adv. Automation Tech.*, Vol. 1, 1989, pp. 108–111.

[71] **N. C. Singer, W. P. Seering**, "Utilizing Dynamic Stability to Orient Parts", *ASME Journal of Applied Mechanics*, Vol. 54, December 1987, pp. 961–966.

[72] **D. R. Strip**, "A Passive Mechanism for Insertion of Convex Pegs", *International Conference on Robotics and Automation*, Scottsdale, AZ, May 1989, pp. 242–248.

[73] **R. H. Sturges**, "Toward a Rational Workholding Methodology", *IEEE International Conference on Robotics and Automation*, May 1990, pp. 1738–1743.

[74] **S. M. Udupa**, "Collision Detection and Avoidance in Computer Controlled Manipulators", Ph.D. Thesis, California Institute of Technology, Department of Electrical Engineering, 1977.

[75] **K. T. Ulrich**, "Computation and Pre-Parametric Design", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 1043, January 1987.

[76] **Y. Wang**, "On Impact Dynamics of Robotic Operations", PhD Thesis, CMU-RI-TR-86-14, Carnegie Mellon University, Sept. 1986.

[77] **A. Witkin, M. Gleicher, W. Welch**, "Interactive Dynamics", *Proceedings of the ACM*, 1990, pp. 11–21.

[78] **Westinghouse Electric Corporation**, "Design for Assembly Calculator", Westinghouse Productivity and Quality Center, PQC-018, May, 1986.

[79] **D. E. Whitney**, "Historical Perspective and State of the Art in Robot Force Control", *IEEE International Conference on Robotics and Automation*, St. Louis, MO, March 1985.

[80] **D. E. Whitney, R. E. Gustavson, M. P. Hennessey**, "Designing Chamfers", *International Journal of Robotics Research*, Vol. 2, No. 4, Winter 1983.

[81] **D. E. Whitney**, "Quasi-Static Assembly of Compliantly Supported Rigid Parts", *Journal of Dynamic Systems, Measurement, and Control*, Vol. 104, March 1982, pp. 65–77.

[82] **P. H. Winston**, Artificial Intelligence, 3rd. Edition, Addison-Wesley, Reading, MA, 1992.

[83] **F. Zhao**, "Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based on Phase-Space Knowledge", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Technical Report 1385, September 1992.