Analysis and Planning of Planar Manipulation Tasks

Randy C. Brost

January 1991
CMU-CS-91-149

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright ©1991 Randy C. Brost

This work was supported by the National Science Foundation, under contract number DMC-8520475. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the United States Government.

Abstract

This thesis addresses the problem of producing reliable solutions to manipulation tasks. Such tasks are strongly influenced by the task geometry, mechanics, and uncertainty. This thesis addresses these issues by applying the techniques of classical mechanics, and extends these techniques to include task geometry and uncertainty. In particular, the thesis addresses manipulation tasks that involve two rigid polygonal objects interacting in a plane; examples include linear pushing, compliant motion, and placing-by-dropping tasks.

For this class of tasks, the thesis defines a collection of generic algorithms that analyze the kinematic, static, dynamic, and motion-specification aspects of a given task. These algorithms identify a continuous bounded set of actions that will reliably achieve the task goal, despite uncertainty in every physical parameter except object shape. The algorithms perform a kinematic analysis to construct the set of reachable (x, y, θ) task configurations, a static analysis to identify the configurations where equilibrium is possible, a dynamic analysis to identify a set of initial configurations that converge to the goal, and a coordinate transformation to identify a set of commanded motions that will achieve the goal.

The kinematic and static analysis algorithms have been fully implemented, and the dynamic analysis algorithms have been partially implemented. These programs were used to synthesize linear pushing actions, to analyze a part interacting with an orienting fixture, and to synthesize placing-by-dropping actions. A series of physical experiments were performed to test the validity of the programs' physical predictions; no failures were observed in these experiments, some of which included over 1200 trials.

This thesis represents a step toward the application of classical mechanics to general manipulation problems; many open problems remain. The thesis presents a discussion of possible extensions of this work to enhance its generality, as well as a discussion of task domains that appear to require a completely different approach.

This thesis is dedicated to the memory of LeVern Cook,
a man of great love, integrity, and humor.

Acknowledgments

I am incredibly fortunate to have had Matt Mason as an advisor throughout my graduate education. Matt has been a terrific teacher, and a true friend. Matt's deep knowledge of the field has been a great resource, and his commitment to excellence in research has always inspired me to do my best. Many of the ideas presented in this thesis were born during lengthy discussions with Matt; his contribution to this work is immeasurable.

I would also like to thank Tomás Lozano–Pérez, Tom Mitchell, and Allen Newell for their comments and guidance as this work developed, and for their suggestions on ways to improve the document. Their perspective and insight have been extremely valuable.

I would like to thank Bruce Donald, Mike Erdmann, and Ken Goldberg for many excellent discussions during the crucial early stages of this work, and for their comments and encouragement in the time that followed. I am also grateful to Alan Christiansen for many thoughtful discussions during the final stages of the thesis; these played a key role in helping me sort out the main ideas in preparation for writing. Srinivas Akella, Mike Caine, Jessica Hodgins, Jeff Koechling, Kevin Lynch, Larry Matthies, Marc Raibert, and Yu Wang also offered technical insights and encouragement; this thesis would surely be lacking without their contributions.

My graduate school years have been graced with many wonderful friends; I thank you all for the encouragement, support, and fun that we've shared. I'd especially like to thank Sanjiv Singh for encouraging me to pursue robotics in the first place, for his steadfast friendship through some very difficult periods, and for those fantastic spicy chick peas.

I could never have completed this thesis without the abundant love, support, and sound advice that my parents and family have given me throughout the years. Finally, I would like to thank Ellen for the tremendous support she offered throughout this thesis process, for insisting that I take a vacation once in a while, and for her terrific sense of humor.

Table of Contents

Abstract i

Acknowledgments v

Table of Contents vii

Overview ix

Part I: The Main Results

A Problem ... 1

A Proposed Solution ... 5

An Experiment ... 25

What Have We Learned? ... 45

Relationship to Previous Work ... 57

Conclusion ... 68

Part II: Models, Data Structures, and Algorithms

Kinematics 69
- The Nature of Configuration-Space Obstacles ... 69
- Representing Configuration-Space Obstacles ... 96
- Constructing Configuration-Space Obstacles ... 106

Statics 135
- Friction and Planar Forces ... 135
- Representing Sets of Forces ... 150
- Constructing Possible-Equilibrium Configurations ... 158

Dynamics 175
- Constructing V_{state} Using Energy Arguments ... 176
  - Generalizing the Notion of Mechanical Energy ... 178
  - The Pigeonhole Principle ... 180
  - Representing "Puddles" in Configuration Space ... 181
  - The BP_e Algorithm ... 183
- Constructing V_{state} Using Numerical Integration ... 202
  - Representing State-Transition Cones, Expandable Contours, and Strong Backprojections ... 210
  - The BP_i Algorithm ... 221

Formulating Motion Commands 251

Part III: Appendices

1. Constructing Configuration Obstacles for Planar Grippers with Two Fingers 255

2. Finding the Roots of Trigonometric Polynomials 261

3. Configuration Obstacle Feature Equations 264

4. Static Analysis Equations 269

5. The *CO* Algorithm 287
   5.1. Global Convexity 287
   5.2. Incremental Surface Construction 288
   5.3. Pseudo-Code Summary of the *CO* Algorithm 292

6. Test Data 307

Bibliography 309

Overview

This thesis is organized as follows:

Part I describes all of the main results of the thesis. This part introduces a class of manipulation problems involving interacting planar rigid bodies, and proposes a unified approach to their solution. This approach extends the notions of classical mechanics to include complex task geometry, uncertainty, and the presence of task constraints. An experiment is proposed to test the viability of the proposed approach, and the results of the experiment are presented. This part concludes with a discussion of what we can learn from this exercise, and how it might affect future work.

Part II gives an expanded description of the solution proposed in Part I. This solution analyzes a manipulation task by constructing a series of geometric sets that describe the kinematic, static, and dynamic aspects of the task; this part explains the models, data structures, and algorithms used to construct these sets. In order to convey a clear understanding of the nature of these sets and their associated computations, this description is primarily qualitative.

Part III contains information that was omitted from Parts I and II for clarity. This includes tables of equations, tables of experimental data, and additional details of the algorithms and procedures presented earlier. These details are included to make the computational processes described in Part II concrete, and to aid the researcher attempting to reproduce or extend these results.

Part I
The Main Results

A Problem

This thesis begins by assuming that we would like to build systems to automatically manipulate objects. If we could build such systems, then we would have at our disposal machines that could reorganize material in the same way that computers now reorganize information. The possible applications would be manifold. We would like these systems to be as autonomous, flexible, and reliable as possible. Among these, reliability is of primary importance, especially when these systems are utilized in hazardous environments where the cost of failure can be extremely high. So this thesis is motivated by the goal of building reliable autonomous manipulation systems.

Toward that end, let's consider one of the simplest manipulation tasks imaginable: moving an object from its current position to a desired position. This general problem statement is fairly sweeping, including the repositioning of planets, removal of atoms from macromolecules, etc, so we will simplify things by restricting our attention to "reasonable" tasks — situations where the object isn't bolted down, is large enough to be perceived by ordinary sensors, small enough to be moved by available acuation systems, and so on.

Even with these simplifications, we are years away from obtaining a general solution to this problem. Why? There are a number of difficulties. The perception problem is an obvious difficulty, but it is not addressed by this thesis. Even with a good perception system available, the task of moving an object to a desired position is extremely hard.

To see why, consider the task shown in Figure 1. The two objects exist in a planar world, with gravity acting into the plane; this world is analogous to a horizontal tabletop. The goal is to move the two objects into a desired coincidence relationship shown by the dotted outline. The larger object is controlled by our manipulation system, which we will refer to as the "robot." The robot can move the large object into desired positions, and also perceive the location of the smaller object.

An obvious solution to this task is to have the robot measure the position of the smaller object, compute the position of the large object that will produce the desired coincidence relationship, and move the large object to that position. If the robot has perfect sensing and control, then this strategy will succeed.

Unfortunately, no real robot can ever have perfect sensing and control, since no physical quantity can ever be measured with infinite precision. Further, some natural phenomena have inherent uncertainty. For example, the coefficient of friction of two surfaces in contact depends intimately on the microscopic interaction of the molecules on each surface; variations in this interaction may

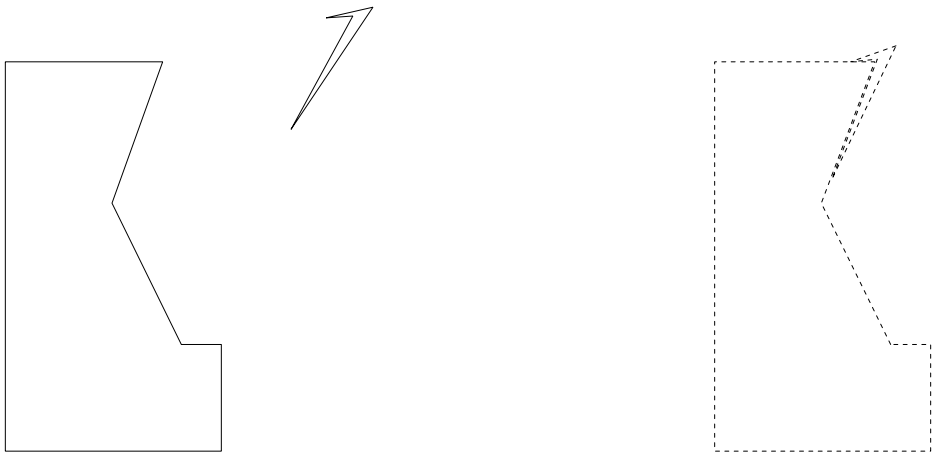


Figure 1: A simple task.

have a signficant effect on the macroscopic behavior of the objects. These considerations imply that the action executed in the world will not exactly match the robot's model of the action, possibly leading to unexpected results (Figure 2).

To attain reliable manipulation systems, we must somehow cope with the presence of uncertainty. There are several approaches to this problem, each of which has been used successfully in the past:

- Choose a task where uncertainty doesn't matter. Examples include spray-painting and spot-welding, where small deviations do not affect the quality of the final result.
- Add sensing and/or fixtures to reduce uncertainty to an acceptable level. This approach is widely applied in modern automation systems, but incurs the disadvantages of increased cost and reduced flexibility. Ultimately this approach fails if the task precision is sufficiently high, as is often the case in assembly applications.
- Execute actions that are inherently robust in the presence of uncertainty. Sometimes the geometry and physics of a task may be exploited to remove uncertainty, or to achieve the desired results despite the presence of uncertainty.

This thesis focuses on the third approach. For example, Figure 3 shows how a pushing strategy may be used to solve the task of Figure 1, despite the presence of uncertainty. The dashed lines show the nominal positions of the objects, and the nominal motion direction. This is the world according to the robot's model; the execution of this action will successfully achieve the goal configuration. However, the true state of the world is slightly different; this is shown by the solid lines. Despite the fact that the true executed motion is noticeably different from the robot's expected motion, the goal is still achieved. This is because the object slips and rotates into the desired position as the pushing operation proceeds; the task mechanics and geometry combine to remove the error.

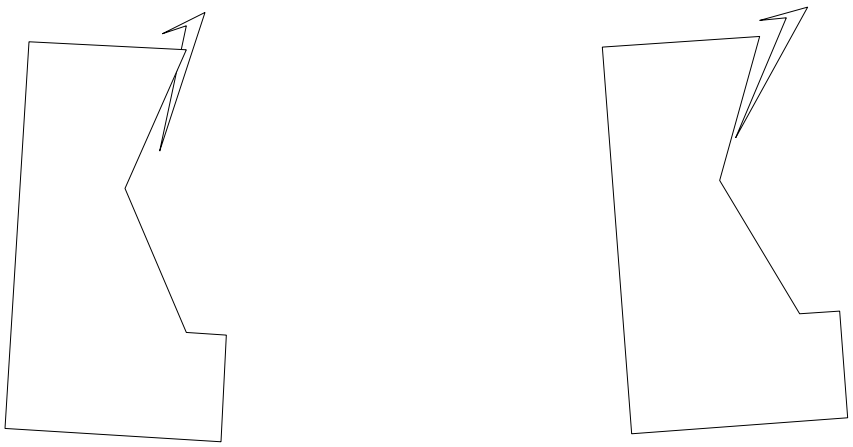


Figure 2: Example failures of the idealized solution due to the presence of uncertainty.

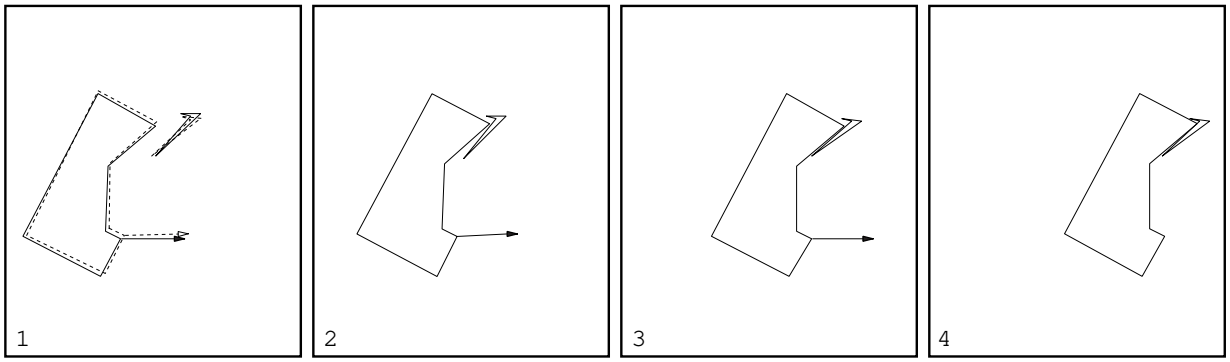


Figure 3: Using a pushing operation to solve the task.

We would like to be able to use actions such as this example pushing motion to solve manipulation tasks in the presence of uncertainty. This gives rise to the question: How do we identify such actions? In general, this will require simultaneous analysis of the task geometry, mechanics and uncertainty.

Some tasks have an additional feature that complicates their analysis and planning, which is the presence of task constraints. A familiar example is the task of moving a glass of water without spilling it. A second example is the task of gluing two parts together; the glue-covered surfaces should never touch anything, except at the final instant when the parts are bonded together. In both of these examples, the task imposes constraints that must be obeyed throughout task execution.

In light of this discussion, we will recast our "simple" task of repositioning an object to include the presence of task uncertainties and constraints. Given an object in the world, a manipulation system, and a desired object position, we would like to identify and execute an action that is robust in the presence of task uncertainties, and obeys task constraints. This thesis addresses this problem, for a special class of objects and task situations.

A Proposed Solution

The previous section discussed the problem of automatically producing reliable manipulation actions; this section will propose a solution to this problem, and show how it may be applied to three different manipulation tasks.

The proposed solution is motivated by two observations about manipulation tasks: First, all manipulation tasks involve physical systems. If we treat the manipulation planning problem as the analysis and control of a physical system, then we can simultaneously improve the physical validity of our solutions, and benefit from advantageous mechanical characteristics of the task. Second, manipulation tasks are embedded in continuous domains. This implies that many manipulation tasks may be solved by a continuous bounded set of similar actions.

These observations suggest the following approach to solving manipulation tasks: Using a model of the physical system behavior, construct a set of actions that will successfully accomplish the task. Then shrink this set by the amount of uncertainty that is present; the actions that remain will accomplish the task despite the presence of uncertainty.

This thesis embraces this idea and makes it concrete, for a restricted class of manipulation tasks that involve two polygonal objects interacting in a planar world. In doing so, the thesis develops a collection of generic algorithms that may be combined in various ways to solve several different manipulation problems. Because these generic algorithms are somewhat abstract, we will defer a formal presentation of their semantics for the moment. Instead, we will develop an intuitive understanding of the algorithms by seeing how they may be used to snythesize the pushing motion shown in Figure 3.

Figure 4 gives a precise definition of this manipulation problem. The task is defined by specifying parameters that describe the objects in the world, their physical properties, their locations, and the performance capabilities of the robot. Except for the object shapes, each parameter has an associated uncertainty, which represents the maximum possible discrepancy between the robot's model of a parameter value and the true parameter value.

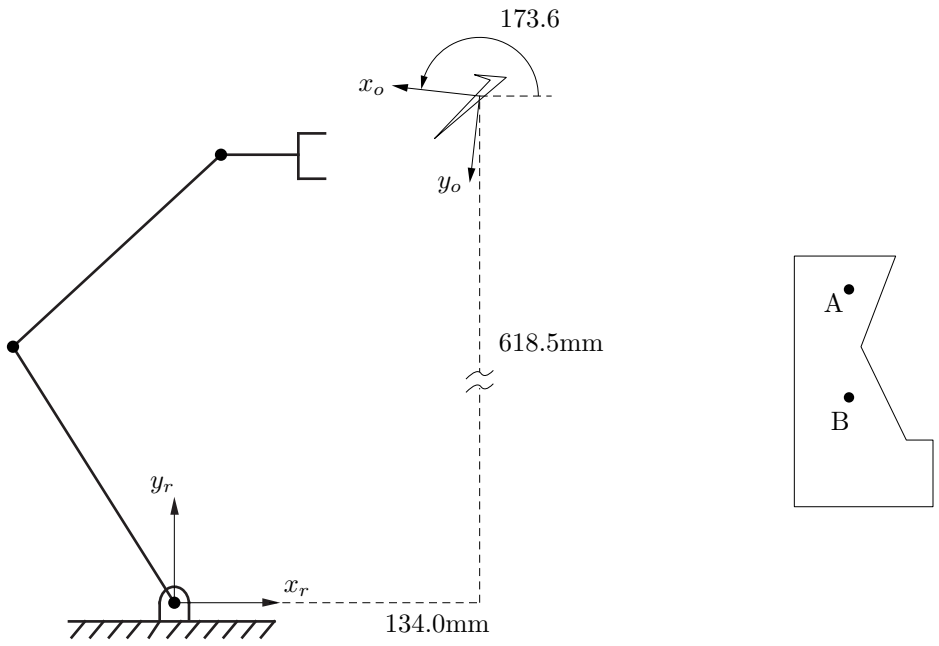


Figure 4: Defining the manipulation task illustrated in Figure 1. The robot believes the small object is at the position (134.0mm,618.5mm,173.6°); the true position is within 2mm and ±5° of this nominal value. The robot believes that the object center of mass is located at its centroid; the true center of mass is within 3.2mm of this point. The distribution of support forces between the small object and the supporting surface is unknown. The robot knows that the true coefficient of friction between the two objects varies within the interval $[0.23, 0.48]$. The robot may grasp the large polygon with one of two handles located at points A and B, and then move the handle to any desired (x, y, θ) location; in doing so, the true resulting position will be within 1mm and ±2° of the commanded position. The robot may also translate the large polygon in any desired direction; the resulting motion will have a true direction that is within ±2° of the commanded direction. The height of the large polygon is 173.7mm.

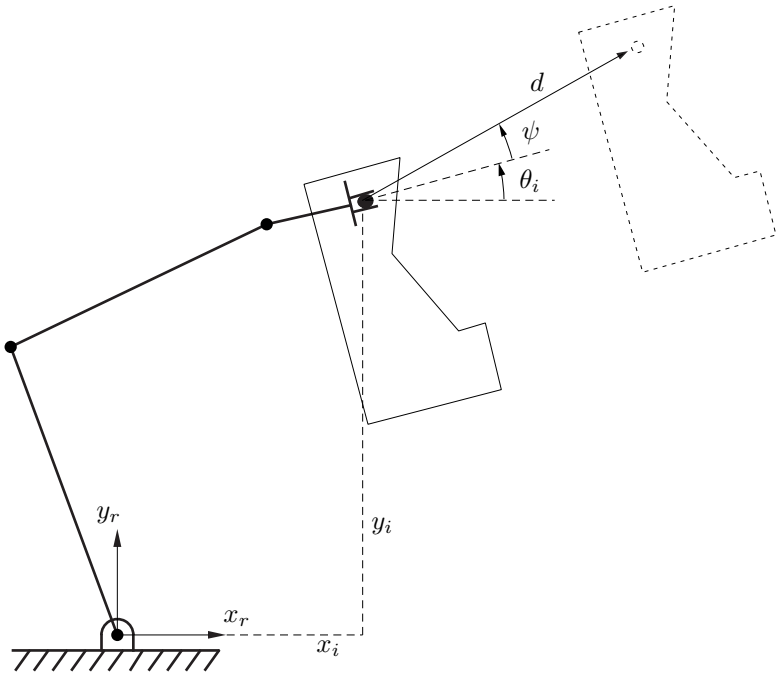


Figure 5: Parameters describing a pushing operation. x_i, y_i, θ_i, ψ, and d are commanded values; control uncertainty will cause the true motion to be slightly different.

We would like to synthesize a pushing action that will achieve the goal shown in Figure 1. A pushing action is executed by grasping the large polygon by one of its two handles, moving it to an initial position (x_i, y_i, θ_i), and then translating in a direction ψ for a distance d. The planning problem then becomes one of choosing which handle to grasp, and identifying values of x_i, y_i, θ_i, ψ, and d that will produce a successful motion (Figure 5). This may be accomplished by the following procedure:

1. Construct the state space of the system.
2. Repeat:
   - 2.1. Propose a pushing direction ψ.
   - 2.2. Construct a set of initial states V_{state} that will converge to the goal under the commanded pushing direction ψ, despite the presence of uncertainty in the task mechanics and pushing direction.
   - 2.3. Transform V_{state} into a set of initial commanded positions V_{command} that will achieve the goal. Include sensing and control uncertainty in this transformation; V_{command} is then a set of commanded initial positions that will achieve the goal, despite the presence of uncertainty in mechanics, sensing, and control.

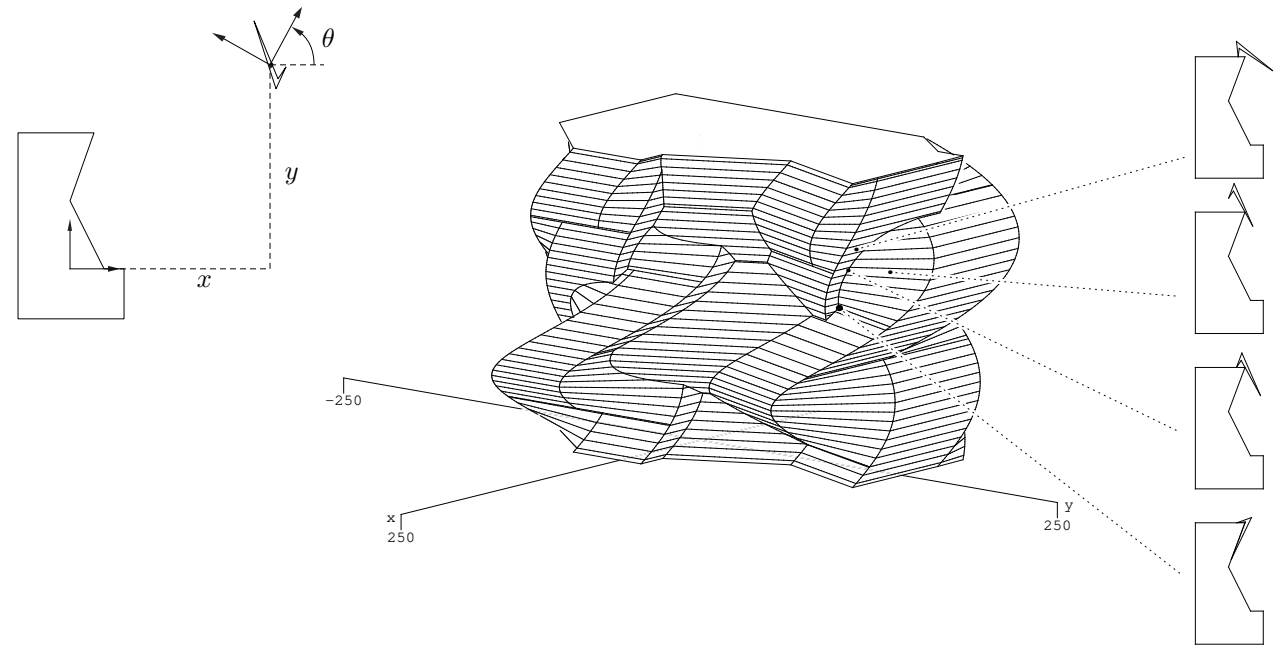
     Attempt this computation for handle A, and if the result is null, then try again for handle B. Set H to a flag indicating whether V_{command} corresponds to handle A or B.

   until V_{command} is non-null, or all candidate ψ values have been exhausted.
3. If V_{command} is non-null, then:
   - 3.1. Select a point (x_i, y_i, θ_i) from V_{command}.
   - 3.2. Compute the required pushing distance d.
   - 3.3. Return the vector $[H \;\; x_i \;\; y_i \;\; \theta_i \;\; \psi \;\; d]$.

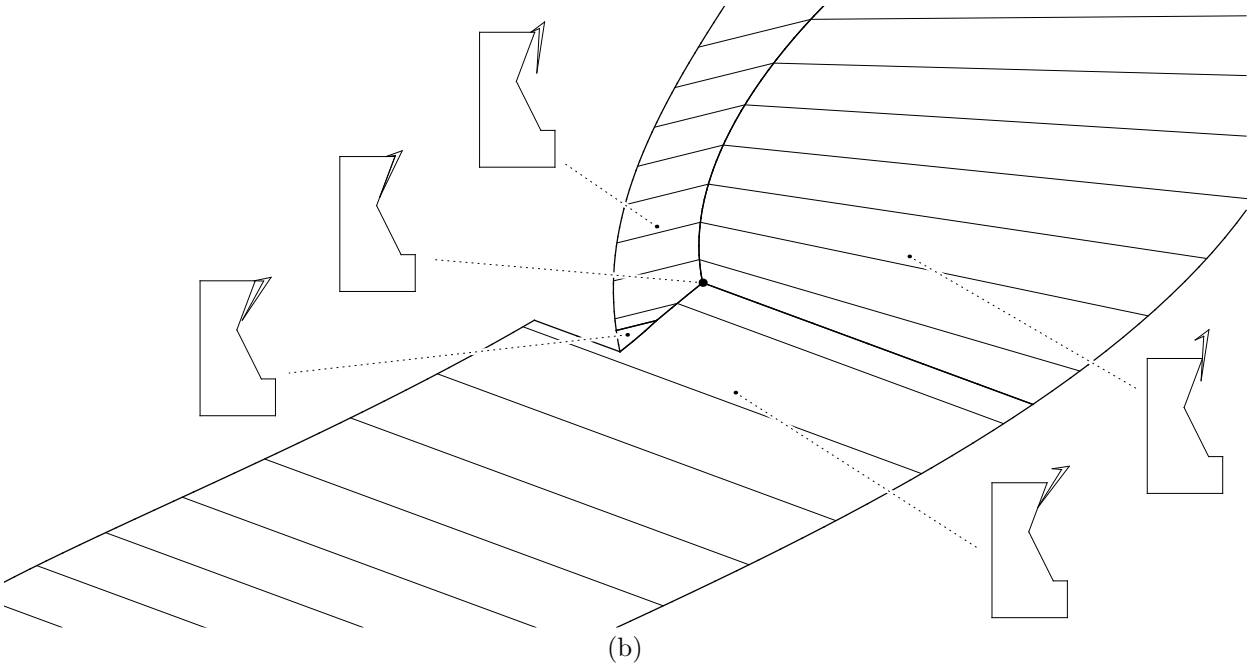
This procedure reveals several weaknesses of the synthesis process as it is currently developed. For example, the pushing direction ψ is not directly constructed, but instead is identified by a generate-and-test strategy, possibly guided by heuristics. In addition, we shall see later that the machinery required to compute the pushing distance d in step 3.2 has not been developed. Finally, there are several important factors that we neglect: We do not consider the workspace limits of the robot, the problem of planning a motion for the robot to grasp the large polygon, or the problem of planning a collision-free motion to move the large polygon to the initial position. Despite these omissions, the above algorithm is able to synthesize a solution to our example pushing task, as well as other more difficult problems. We shall examine steps 1, 2.2, and 2.3 in further detail.

Step 1: Constructing the State Space

We begin with a kinematic analysis, shown in Figure 6. The surface depicted in the figure is the *configuration-space obstacle* of the two interacting polygons. This surface is embedded in the (x, y, θ) space of possible configurations; the horizontal axes are the x and y axes, and the vertical axis is the θ axis. Points inside the configuration-space obstacle correspond to illegal positions where the objects overlap, while points outside the obstacle correspond to free positions where the



(a)



(b)

Figure 6: (a) The (x, y, θ) configuration-space obstacle of the two polygonal objects. (b) An enlarged view of the obstacle features required for the mechanics analysis. The vertex corresponding to the goal configuration is labelled in both (a) and (b).

objects do not touch. Points on the obstacle surface are positions where the objects are in contact. On this surface, we can identify features that are analogous to the faces, edges, and vertices of a polyhedron. These features correspond to sets of configurations that have identical topological contact conditions; a few of these configurations are labelled in the figure.

The configuration-space obstacle provides an exact description of the kinematic properties of the two interacting polygonal objects. The surface of the obstacle describes all of the possible states where the objects are in contact, and the free-space surrounding the obstacle corresponds to all of the possible states where the objects are not in contact. Thus the combination of these two sets gives a complete description of the possible kinematic states our physical system can attain. This is a sufficient state space for our analysis, because we will not require velocity information.

Step 2.2: Constructing a Set of States that Converges to the Goal

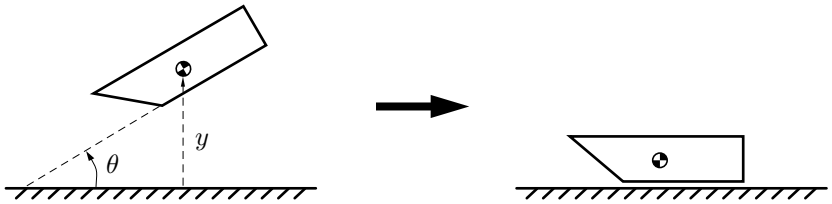
Given a goal on the surface of a configuration-space obstacle and a pushing direction, we would like to identify a set of initial configurations that will converge to the goal as the pushing operation is executed. Here we present two methods for constructing such a set. In the first method, we assume a weak model of the task mechanics, and apply energy arguments to construct the set. In the second method, we assume a stronger model of the task mechanics, and construct the set through numerical integration. Both methods compute a conservative approximation of the true set of initial configurations that will reliably converge to the goal, but the second method is less conservative than the first.

Method 1: Energy Arguments

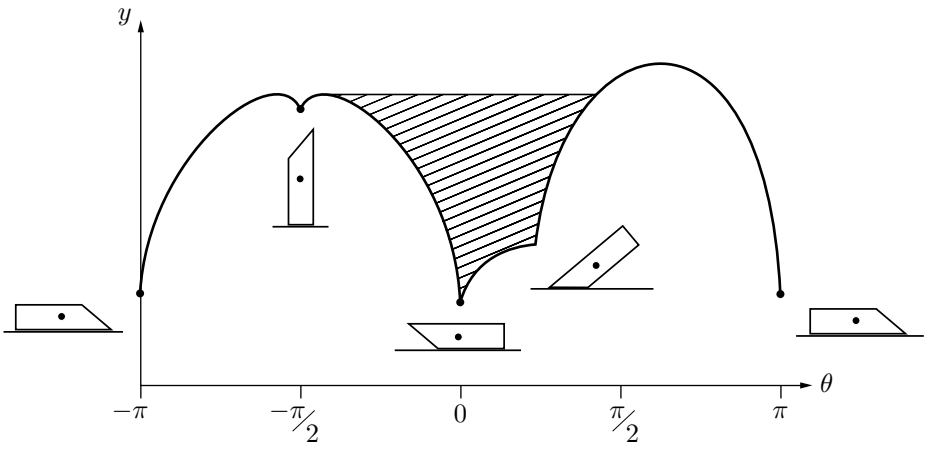
In this method, we combine the kinematic constraints of the configuration-space obstacle with "energy" constraints imposed by the task mechanics to identify a set of initial configurations that must converge to the goal. Figure 7 shows how this is performed in a simple two-dimensional configuration space. In this task, the goal is to place the polygon on the flat surface in the configuration shown. We will neglect the lateral position of the polygon, so the relevant configuration parameters are just y and θ. The corresponding configuration-space obstacle is shown in the figure; points below the bold curves are infeasible configurations.

We would like to synthesize a dropping motion that will reliably place the polygon in the desired configuration, which corresponds to one of the deep concavities in the configuration-space obstacle. A simple way to accomplish this is by observing that if there is no initial angular velocity at the time of release, then the mechanical energy of the system is bounded by the potential energy of the polygon's initial release position. A consequence of this observation is that the trajectory of the system in the configuration space is constrained to stay below the height of initial release.

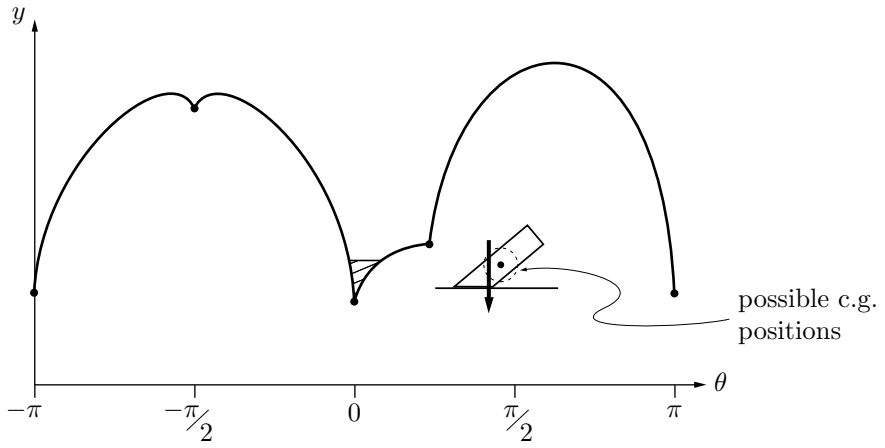
We can exploit this property to identify initial release positions that will converge to the goal. The horizontal line in Figure 7(b) shows an initial release height that constrains the possible configurations to only one of the "valleys" in the configuration space; this valley contains the goal configuration. If the polygon is dropped from an initial configuration within the shaded region, then we know that it can never escape the region, since this would require an increase in mechanical energy. Further, we can use a separate analysis to show that the only possible rest configuration in the valley is the goal; thus if the system ever comes to rest, then it must come to rest in the goal. Unless the polygon is perfectly elastic and frictionless, it will lose energy every time it contacts the surface, and eventually come to rest. Thus, if the polygon is released from a point within the region, it will come to rest in the goal configuration.



(a)



(b)



(c)

Figure 7: Using energy arguments to synthesize dropping motions. (a) The goal. (b) The configuration space. The shaded region corresponds to a set of initial release positions that will achieve the goal. (c) Adding mass uncertainty creates a new stable state, which reduces the set of release positions guaranteed to achieve the goal.

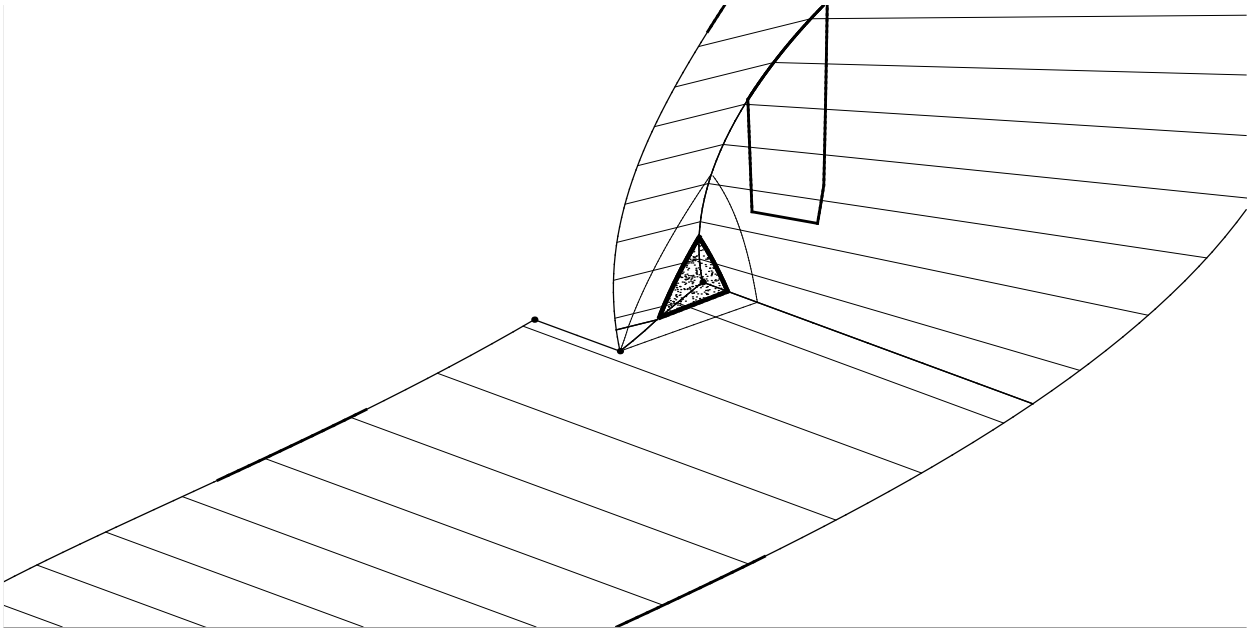


Figure 8: A set of configurations that will achieve the goal for $\psi = 29° \pm 4°$. The shaded region is the floor of a volume of configurations that is a "puddle" on the obstacle surface. Removing uncertainty in the pushing direction and center of mass location would result in a larger volume, bounded by the light contour surrounding the shaded region. The other highlighted features represent the configurations where equilibrium is possible; since these regions are far from the no-uncertainty contour, they did not restrict the volume in this example.

The existence of other possible rest states may reduce the convergence region. In Figure 7(c), there is uncertainty in the position of the center of mass, which makes static equilibrium possible for the configuration labelled in the figure. Since this is not a goal configuration, we must preclude this possible final rest state by choosing a more restrictive initial release height. In addition, this height must be further reduced to assure that the energy is sufficiently restricted for all possible locations of the center of mass.

This example shows how energy arguments may be combined with configuration-space constraints and a static-equilibrium analysis to identify a set of configurations that will converge to a desired goal. Notice that the analysis makes no assumption about the details of the polygon's trajectory as it bounces before coming to rest; these collisions may be arbitrarily complex without invalidating the result. Further, these arguments are not specific to tasks involving gravity; they may be applied whenever there is an appropriate metric in the task mechanics that bounds the set of reachable configurations.

Figure 8 shows the result of applying this approach to our example pushing task. A nominal pushing direction has been chosen by hand, and this direction imposes a metric in the task that is analogous to potential energy.[1] We can use this metric to identify a constraint that bounds the set of reachable configurations to a concavity which contains the goal, but no other possibly-stable configurations. The resulting volume of initial configurations is shown in the figure; this volume is analogous to the puddle that would form if you poured water onto the obstacle surface, with gravity pointed in the direction imposed by the "energy" metric.

[1]This is actually a conjecture, and will be discussed further in Part II.

Method 2: Numerical Integration

The energy-based method described above provides a direct technique for computing a set of initial states that will converge to the goal. Unfortunately, there are some situations where this approach is not sufficient; these situations include cases where the goal is not locally concave, cases where the goal is adjacent to non-goal states where equilibrium is possible, and cases where there is so much uncertainty that the constructed volume vanishes after shrinking. In these situations, we must apply a more powerful technique to identify a set of convergent initial conditions.

To accomplish this, we will apply a stronger model of the task mechanics that includes trajectory information. By including more information describing the system behavior, we can compute a less conservative approximation of the true set of initial configurations that will achieve the goal.

We do not require that our trajectory model produce a unique prediction of the system behavior; in fact, this would be a very unrealistic expectation for models of mechanical systems in the presence of uncertainty. Instead, we use a trajectory model that provides a description of the *set* of possible system behaviors consistent with a given configuration and a given action.

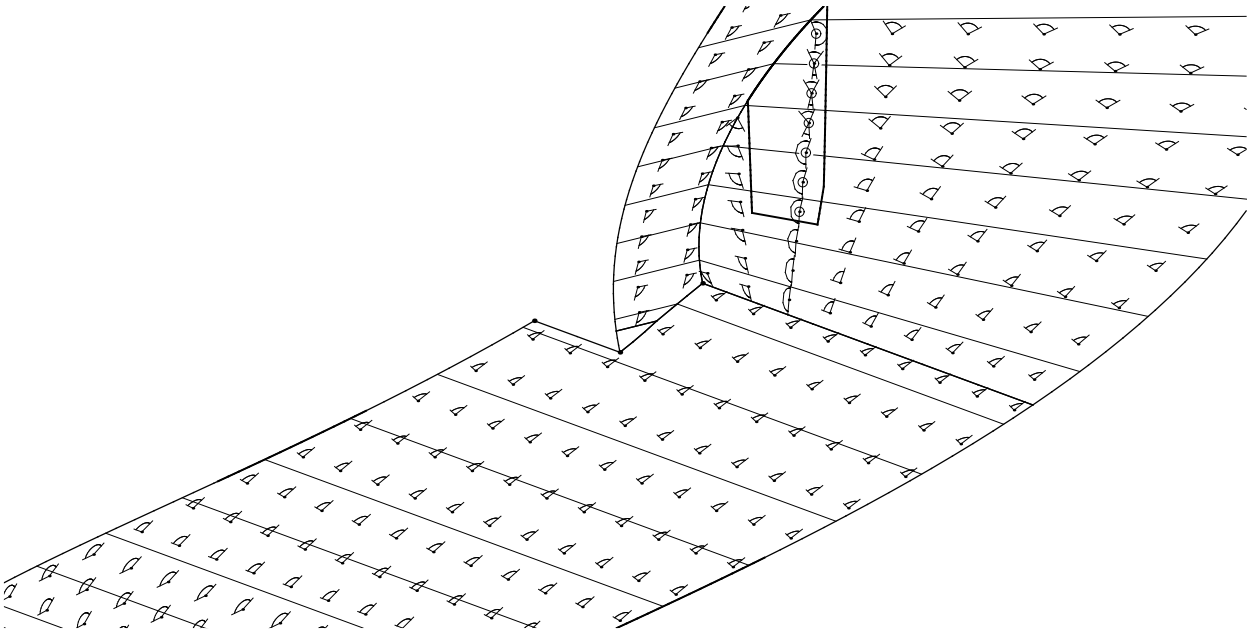


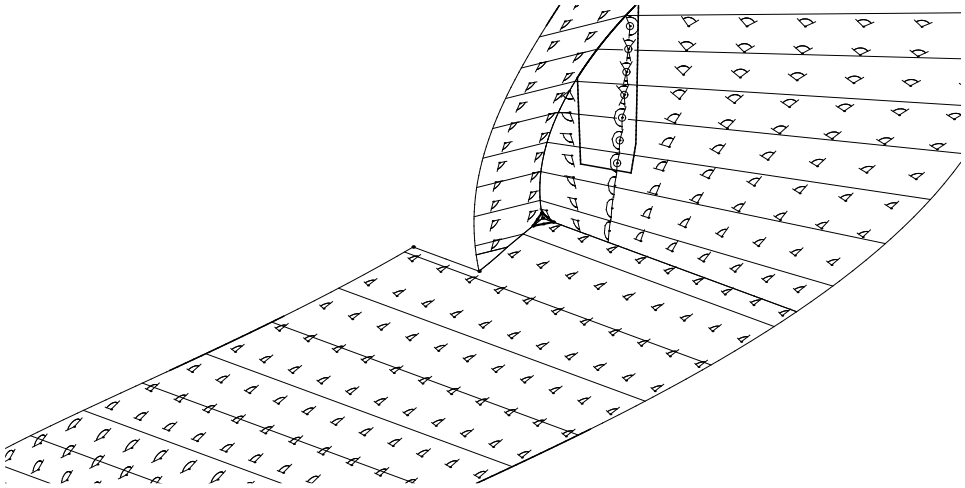
Figure 9: Pushing mechanics expressed in the configuration space. Each cone depicts the set of possible instantaneous motions that may occur for $\psi = 29° \pm 4°$. For a configuration corresponding to the base of a given cone, the configuration that will occur at the next time instant will lie somewhere within the angular limits indicated by the cone's arc. A circle around the base of a cone indicates that zero-motion is possible; notice that these cones only exist within the possible-equilibrium contour computed by other methods. This contour also contains three double cones, indicating that for those configurations, rotation in either direction is possible, but pure lateral sliding is not possible. Lost-contact is also a possible case, but no example appears in this figure. These cones include the effect of uncertainty in mass, support friction, contact friction, and pushing direction.

Figure 9 shows the trajectory model for our example pushing task. Due to the uncertainty in mass, support friction, contact friction, and pushing direction, the exact motion of the object cannot be predicted. However, it is possible to constrain the set of possible motions, and these constraints may be expressed by a cone of possible instantaneous motions. The figure shows these cones for a sampling of points on the obstacle surface; note that multiple cones appear for some points, as well as situations where zero motion can occur.

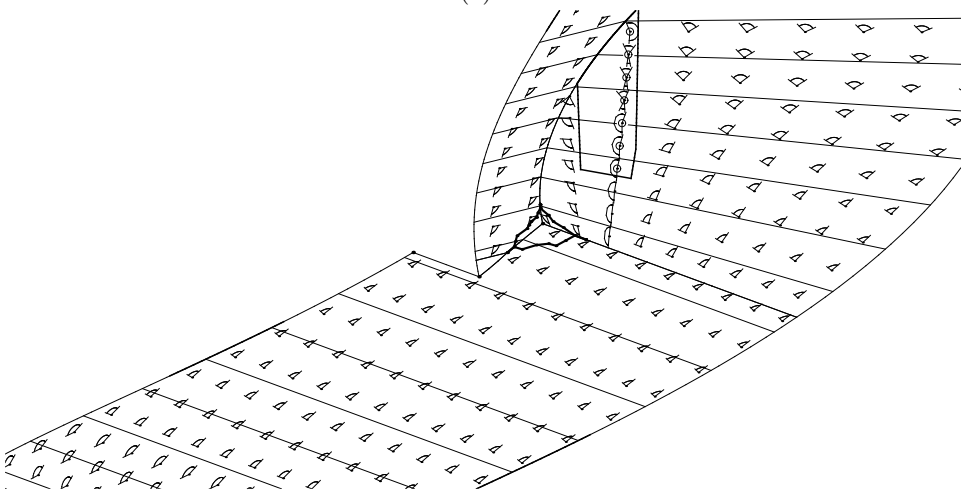
We can use this trajectory model to compute a region of initial contact states that will converge to the goal. This is accomplished by a numerical procedure that constructs an initial contour surrounding the goal, and then expands this contour to include points that will converge to the goal. Conceptually, this is a greedy algorithm: The initial contour contains only goal points, and then the algorithm considers each adajcent point outside the contour and asks the question: Do all of the possible trajectories from this point enter the current contour? If the answer is yes, then the contour is expanded to include the new point, and the process repeats. Thus the initial contour expands outward, but at all times the contour only includes points that will achieve the goal. This expansion continues until no more points may be included, or until some other termination criterion is met.

Figure 10 shows this process applied to our pushing example. We begin with an initial contour surrounding the goal, and gradually expand this contour until no further expansion is possible. The resulting contour bounds a region of configurations on the obstacle surface that will maintain contact and converge to the goal. We can convert this two-dimensional set into a volume by identifying the free positions that give rise to initial contacts within the contour. This is accomplished by erecting the extremal pushing direction rays from every contour point; these rays bound a volume of initial configurations that will achieve the goal. If the rays re-intersect the obstacle surface, then we must construct a new contour surrounding the intersection and repeat the contour expansion/ray construction process. This does not occur in this example.

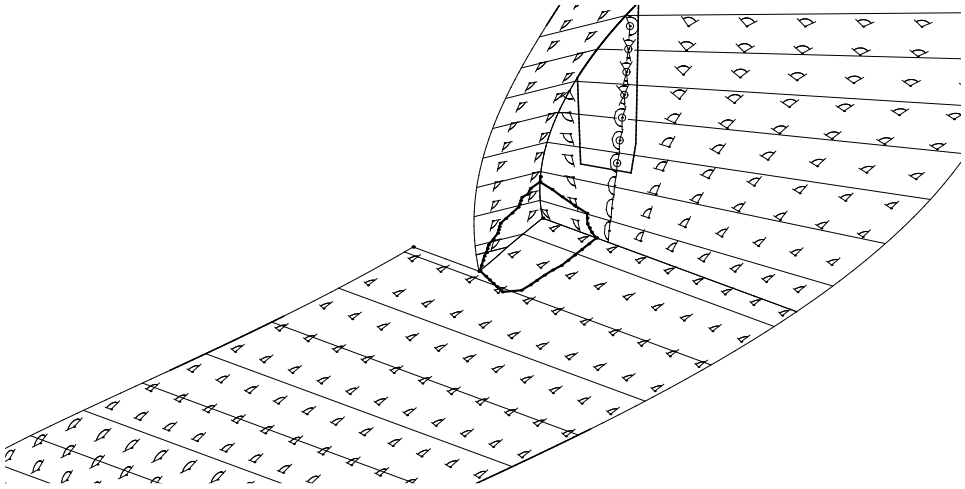
A comparison of figures 10(e) and 8 shows an advantage of applying a more detailed model of the task mechanics: The volume constructed with the trajectory model is significantly larger than the volume constructed using only energy constraints. This is a consequence of the additional constraints imposed by the trajectory model, which restrict the trajectories that are possible on the obstacle surface and in free space. These constraints allow the algorithm to identify a larger bounding contour on the obstacle surface, and a deeper volume for a given contour.



(a)

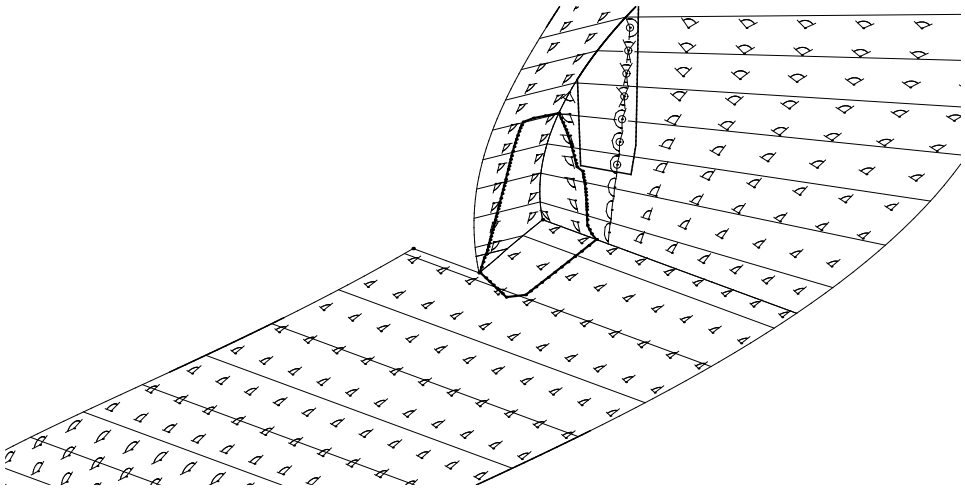


(b)

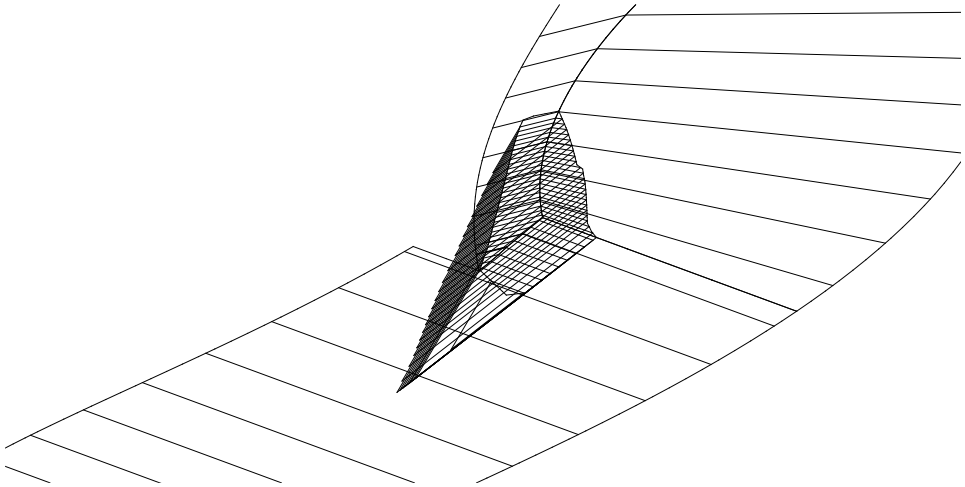


(c)

Figure 10: Constructing a volume of initial states that will achieve the goal, using numerical integration. (a) The initial contour surrounding the goal. (b) After 3 expansion steps. (c) After 10 expansion steps.



(d)



(e)

Figure 10 (continued): (d) After 36 expansion steps; no further expansion is possible. At all times during the expansion process, the trajectory cone for each point on the contour points strictly into the contour. (e) Converting the two-dimensional region into a volume, using the minimum and maximum pushing direction rays. This figure depicts a discrete approximation of the ideal continuous algorithm.

Step 2.3: Transforming Initial States into Commanded Starting Positions

We have seen two methods for computing a set of initial states V_{state} that will achieve the goal configuration for a pushing direction ψ, despite the presence of uncertainty in task mechanics and pushing direction. Since we ultimately seek a robot pushing command that will achieve the goal, we need to convert this set of initial states into a set of commanded starting positions. We accomplish this by transforming V_{state} into the robot's coordinate system; in addition, we apply shrinking operations to compensate for the position uncertainty of the object and the robot. These operations are performed by the following procedure:

2.3.1. Shrink V_{state} by the uncertainty in the object position.

2.3.2. Transform the shrunk volume into the robot's coordinate frame; this transformation depends on whether the robot uses handle A or B to grasp the large polygon.

2.3.3. Shrink the transformed volume by the robot's position uncertainty.

The order of these operations is significant; we must compensate for the object's position uncertainty before applying the coordinate transformation, and the robot's uncertainty after applying the transformation. Enforcing this order ensures that the shrinking procedures are performed in the proper coordinate systems — the shrinking for object position uncertainty is performed in the (x, y, θ) space of relative object positions, and the shrinking for robot position uncertainty is performed in the (x_i, y_i, θ_i) space of commanded starting positions.

Figure 11 illustrates this procedure for our pushing example. The volumes for handles A and B are both shown in the figure; notice that the volume for handle B vanishes after shrinking for the robot position uncertainty, even though the original volume and uncertainty parameters were the same in both cases. Why?

This can be understood by considering the *center of uncertainty* of a tool being applied to a task. If we say that the tool may be moved to a position (x, y, θ) with a precision of ϵ_{xy} distance units and ϵ_θ degrees, then the center of uncertainty is the point on the tool where these precision limits are defined to hold. Other points on the tool will have larger effective position errors, because the angular error is multiplied by the distance to the center of uncertainty.

For example, suppose that you are asked to insert a meter stick into a small hole in a wall. Where do you grasp the meter stick? One option is to grasp the stick at its midpoint, thus supporting the center of mass. However, this is a bad choice of grasp positions, because the half-meter distance to the tip of the stick would greatly increase the magnitude of small angular errors in your wrist, making the task more difficult. A better choice would be to grasp the stick near the tip, possibly using an additional hand to support the weight of the stick if necessary. This strategy has the effect of moving the center of uncertainty nearer to the tip of the meter stick, which is the critical point for the insertion.

In our pushing example, the robot controls the position of its end-effector, so this defines the center of uncertainty. This point may be collocated with either handle A or B. Handle B is farther away from the upper-right corner of the large polygon than handle A, so angular errors affect handle B more than handle A. This effect becomes apparent once the volume V_{state} is transformed into the robot's coordinate space; the volume for handle B is "smeared" over a large xy-range, which causes the volume to vanish once shrinking for θ-uncertainty is performed.

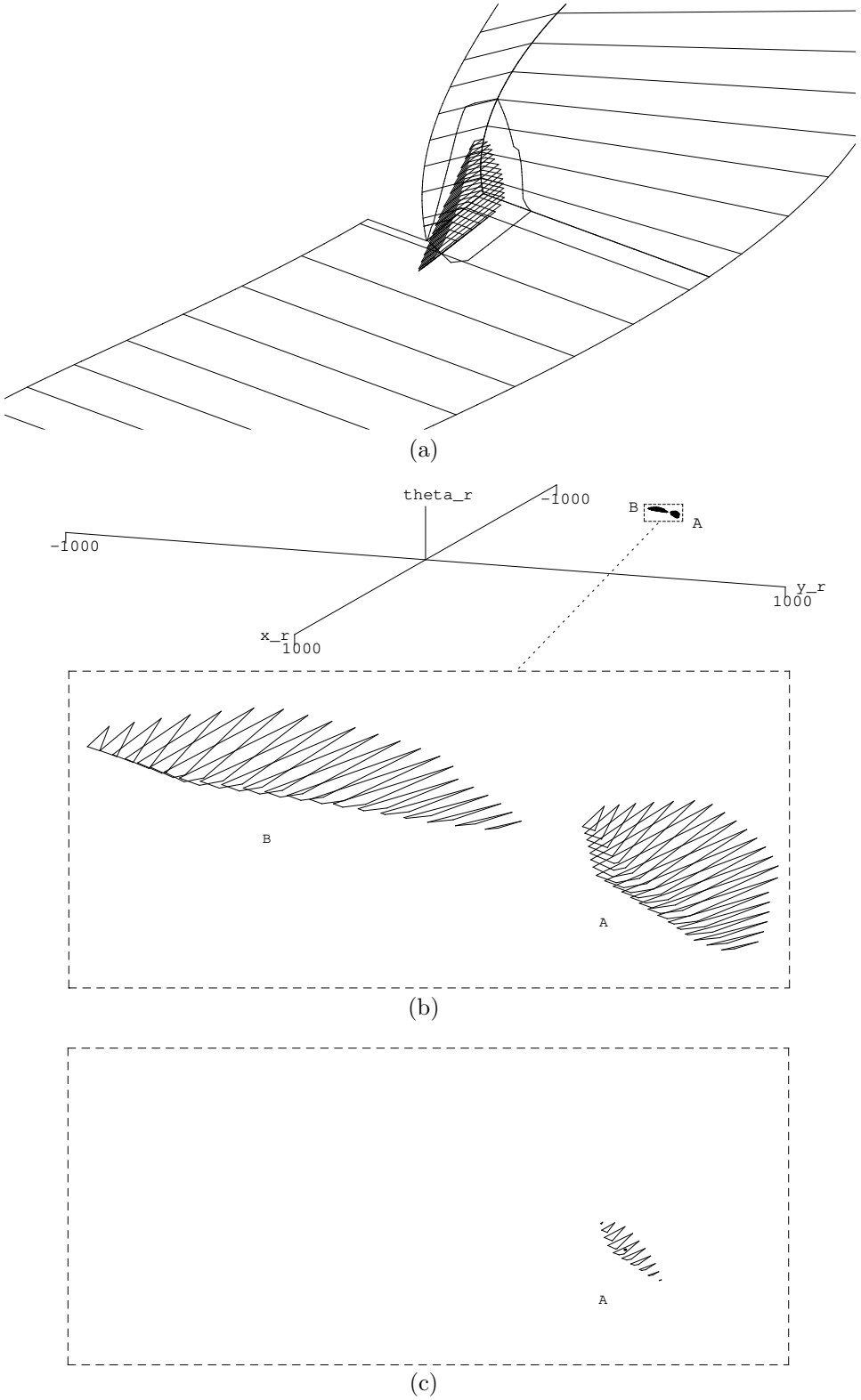


Figure 11: Converting the volume of successful initial states into a volume of successful robot starting positions. (a) After shrinking for object position uncertainty. (b) After converting the volume into the robot's coordinate system. The labels A and B indicate the handle used by the robot to grasp the large polygon. (c) After shrinking for robot position uncertainty. Volume B has vanished. The dot in the center of the volume corresponds to the commanded initial position shown in Figure 3.

The volume for handle A does not vanish. The remaining volume corresponds to a set of commanded starting positions from which a subsequent push in the commanded direction ψ will be guaranteed to achieve the desired final configuration, despite the presence of uncertainty in task mechanics, sensing, and control. The pushing motion illustrated in Figure 3 represents a point near the center of this volume; this point is shown in Figure 11(c).

Generalizing to Other Tasks

The preceding sections showed how a series of set constructions may be used to synthesize robust pushing actions in the presence of uncertainty. We would like to generalize these constructions to cover a variety of planar manipulation tasks. Toward this end, this thesis separates these constructions into a collection of five generic algorithms, which may be combined in various ways to solve several different manipulation problems. These algorithms are briefly summarized below:

CO	Constructs the configuration-space obstacle of the two objects.
STATIC	Constructs the set of configurations where static equilibrium is possible.
BP_e	Constructs a volume of configuration-space points that will converge to the goal, using energy arguments.
BP_i	Constructs a volume of configuration-space points that will converge to the goal, using numerical integration.
COMMAND	Converts a configuration-space volume into a set of commanded robot positions.

These algorithms are analogs to classical techniques commonly used in the analysis of mechanical systems. *CO* analyzes the system kinematics, while *STATIC* addresses the static properties of the system. BP_e and BP_i address the dynamic system behavior, and *COMMAND* converts the analysis results into a form suitable for robot execution. The algorithms are not specific to a particular physical system, but rather apply to any physical system that satisfies the required assumptions. These assumptions and the input/output semantics of each algorithm are presented in Figure 12.

These generic algorithms may be applied to a variety of manipulation problems, as long as the underlying assumptions are satisfied. We conclude this section by discussing the application of these algorithms to three example problems: synthesis of linear pushing actions, the analysis of a fixture designed to orient parts, and synthesis of placing-by-dropping actions.

Linear Pushing

Figures 6-11 show how these algorithms may be used to synthesize pushing motions. We can use the *CO* algorithm to compute the configuration obstacle, and then identify a volume of successful initial states by applying the BP_i algorithm, using a D function based on the analysis of linear pushing mechanics described in Part II. We can also use the *STATIC* and BP_e algorithms to compute the volume, provided that a certain conjecture proposed in Part II is true. Regardless of the method used, the resulting volume of configuration-space points may be converted into a set of successful commanded starting positions using the *COMMAND* algorithm.

Fixture Analysis

Many industrial assembly lines contain machines that are designed to orient parts into a known configuration. Often these machines incorporate special fixtures that are designed to hold the part in only a finite number of possible configurations; the randomly-oriented part is placed into the fixture, and then mechanical filters are applied to remove undesired configurations. Rejected parts are returned to the parts bin to repeat the process. With suitably designed fixtures and mechanical filters, these machines can be very effective in producing a continuous stream of properly oriented parts.

These machines are currently designed by a manual trial-and-error process; this process may be aided by the *CO* and *STATIC* algorithms. Using these algorithms, we can identify all of the configurations where a part may come to rest in a given fixture. This information may be used to confirm that the fixture can hold the part in the desired configuration, and also to identify the undesired rest configurations that must be removed with mechanical filters. Further, examination of these configurations may suggest changes to the fixture design that will reduce the number of undesired rest states, thereby reducing the number of filters required.

Placing by Dropping

If a robot has the goal of placing an object in a desired configuration, one strategy is to drop the object into place. Depending on the details of the task geometry and other parameters, it may be possible to drop the object into the desired configuration from a broad set of initial dropping positions. We can use the *CO, STATIC,* BP_e, and *COMMAND* algorithms to synthesize a set of commanded dropping positions that will achieve the desired final configuration, despite the presence of uncertainty.

$CO(\mathcal{P}_m, \mathcal{P}_f)$

Input: Two polygons, a "moving-polygon" $\mathcal{P}_m$ and a "fixed-polygon" $\mathcal{P}_f$.

Output: A data structure representing the configuration-space obstacle of the two polygons, expressed in the fixed-polygon's coordinate frame. The data structure contains metric, topological, and contact information describing each obstacle feature.

Assumptions: $\mathcal{P}_m$ and $\mathcal{P}_f$ are rigid, and may be comprised of multiple rigidly-connected polygons.

All motions of the system are confined to the plane.

$STATIC(\mathcal{P}_m, \mathcal{P}_f, \mathcal{T}_\mu, \mathcal{F}_a)$

Input: Two polygonal objects $\mathcal{P}_m$ and $\mathcal{P}_f$. Each polygon feature has an associated material.

A table $\mathcal{T}_\mu$ of friction coefficients, indexed by pairs of materials. Each entry in the table should contain an interval $[\mu_{\min}, \mu_{\max}]$ of possible friction coefficients.

A set of possible applied forces $\mathcal{F}_a$, specified by:

- An interval $[\eta_{\min}, \eta_{\max}]$ of possible force-direction angles, measured relative to the fixed-polygon's x-axis.
- A nominal point of application P_η of the applied force, expressed in the moving-polygon's coordinate frame.
- An error distance r_η.

Output: A data structure representing the set of configurations where static equilibrium is possible; these are the configurations where the contact reaction force can balance the applied force.

Assumptions: All of the assumptions of the *CO* algorithm.

Friction forces obey Coulomb's Law: $F = \mu N$.

When the system is in static equilibrium, the relative applied force satisfies the following three conditions: The force has non-zero magnitude, the force direction lies in the interval $[\eta_{\min}, \eta_{\max}]$, and the line of force passes within r_η of the point P_η. The relative applied force is the sum of all forces that are not contact reaction forces.

$BP_e(\mathcal{P}_m, \mathcal{P}_f, \mathcal{T}_\mu, \mathcal{F}_a, \mathcal{G}, \mathcal{C})$

Input: The input parameters $\mathcal{P}_m$,$\mathcal{P}_f$,$\mathcal{T}_\mu$, and $\mathcal{F}_a$ as defined by the *STATIC* algorithm.

A set of desired goal configurations $\mathcal{G}$.

A set of task constraints $\mathcal{C}$, defining a set of configurations that should never occur.

Output: A set of configurations V_{state} which will eventually reach a stable configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{C}$.

Assumptions: All of the assumptions of the *STATIC* algorithm.

The state of the system is bounded by a constraint analogous to the total mechanical energy of a body released from rest in a gravitational field. That is, there is a direction $\eta_{\text{true}} \in [\eta_{\min}, \eta_{\max}]$ and a point P_{true} within a distance r_η of the point P_η such that the system may only reach configurations within the half-space defined by a line normal to η_{true} and passing through P_{true} in the initial state of the system.

The system is not perfectly elastic or perfectly frictionless.

Figure 12: Semantics of the generic algorithms.

$BP_i(\mathcal{P}_m, \mathcal{P}_f, D(x, y, \theta, \text{feature}), \mathcal{G}, \mathcal{C}, \textit{stop?})$

Input: Two polygonal objects $\mathcal{P}_m$ and $\mathcal{P}_f$, with definitions of the properties required by the function D, and appropriate uncertainties in those properties.

A function D that accepts a configuration (x, y, θ) and a configuration obstacle feature, and returns the set of all possible instantaneous motions, along with a Boolean flag indicating whether static equilibrium is possible. The returned set should contain a set of velocity cones describing the motions that can occur in free-space and in contact with the obstacle surface.

A set of desired goal configurations $\mathcal{G}$.

A set of task constraints $\mathcal{C}$, defining a set of configurations that should never occur.

A termination predicate *stop?* that may be used to interrupt the integration procedure and cause it to return its current result; this predicate is explained in Part II.

Output: A set of configurations V_{state} which will eventually reach a stable configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{C}$. Additional output is returned that allows execution to continue if the *stop?* predicate was used to interrupt integration.

Assumptions: All of the assumptions of the *CO* algorithm.

The function D computes a set that includes all possible instantaneous motions, without requiring a description of the object velocities. Under the current algorithm, all free-space motions must be pure translations in a direction within an interval $[\eta_{\min}, \eta_{\max}]$.

The system behavior represented by the function D is well-behaved over small distances within a given configuration-obstacle feature; this assumption is discussed in Part II.

$COMMAND(V_{\text{state}}, \mathcal{U}_{\text{object}}, \mathcal{U}_{\text{robot}}, \text{moving?})$

Input: A set of (x, y, θ) configurations V_{state}.

The object and robot position uncertainties $\mathcal{U}_{\text{object}}$ and $\mathcal{U}_{\text{robot}}$. Each of these is defined by a center of uncertainty P_ϵ and maximum position and orientation errors ϵ_{xy} and ϵ_θ.

A boolean flag "moving?" that indicates whether the robot controls the moving-polygon or the fixed-polygon.

Output: A set of commanded positions V_{command} such that if the robot moves to a commanded position in V_{command}, the resulting true (x, y, θ) configuration will be contained in V_{state}.

Assumptions: All of the assumptions of the *CO* algorithm.

Figure 12 (continued): Semantics of the generic algorithms.

An Experiment

The previous sections presented the problem of planning reliable manipulation actions in the presence of uncertainty, and proposed a collection of five generic algorithms that may be used to solve a restricted class of these problems. These algorithms treat the manipulation task as a physical system, and model uncertainty by associating a worst-case error bound with every physical parameter except the object shape. Using these models, the algorithms explicitly construct a set of action commands that will succeed in accomplishing the goal, despite the worst-case combination of errors.

While the specification of these algorithms may aid our understanding of the structure of manipulation tasks, there is a vast chasm between a theoretical specification and an effective working system. This thesis asks the question: Can this approach be made practical? There are several issues that immediately come to mind:

- Can the necessary analytical details be worked out?
- Can the algorithms be implemented?
- Will the resulting programs be robust?
- Will the resulting programs be reasonably fast?
- Will the plans constructed by the algorithms succeed in the real world?
- Will the algorithms' worst-case analysis be too conservative?

This thesis may be viewed as an experiment that attempts to find answers to these questions. The experimental procedure is simple: Perform the necessary analysis, implement the algorithms, and measure their performance. Successful results will provide a lower bound on the performance that may be achieved using these techniques; lack of success will be less instructive, since more work may lead to improvements.

Work on this experiment is not yet complete; this thesis reports the results obtained thus far. These results are encouraging, but several issues remain unresolved. This section will summarize these results; detailed test data are given in Part III.

Analysis and Implementation Results

As of this writing, the *CO* and *STATIC* algorithms are essentially complete; all of the required analysis has been performed, the full algorithms have been implemented, and the resulting programs have been carefully tested. In contrast, much work remains for the BP_e, BP_i, and *COMMAND* algorithms, and only partial implementations have been produced.

CO

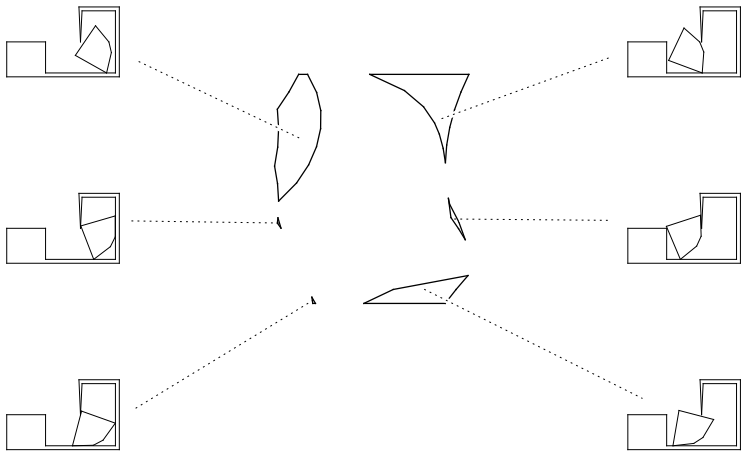
For the *CO* algorithm, an enumeration argument was used to identify all of the configuration-obstacle features that are possible. Once the types of features were identified, a metric and topological analysis was performed. The metric analysis developed parameterized equations describing each feature, and the topological analysis catalogued the possible topological interactions between features. This analysis included both generic and non-generic cases. These results led to the development of an algorithm to construct the obstacle features and their topological arrangement.

This analysis was then used to write a program that constructs a representation of the configuration obstacle, given two arbitrary polygons. Since this is the first step of all of the anticipated manipulation analysis algorithms, substantial effort was devoted to producing a robust and efficient implementation. The resulting program is able to produce some fairly complex configuration obstacles, including the example obstacle features shown in Figure 13.

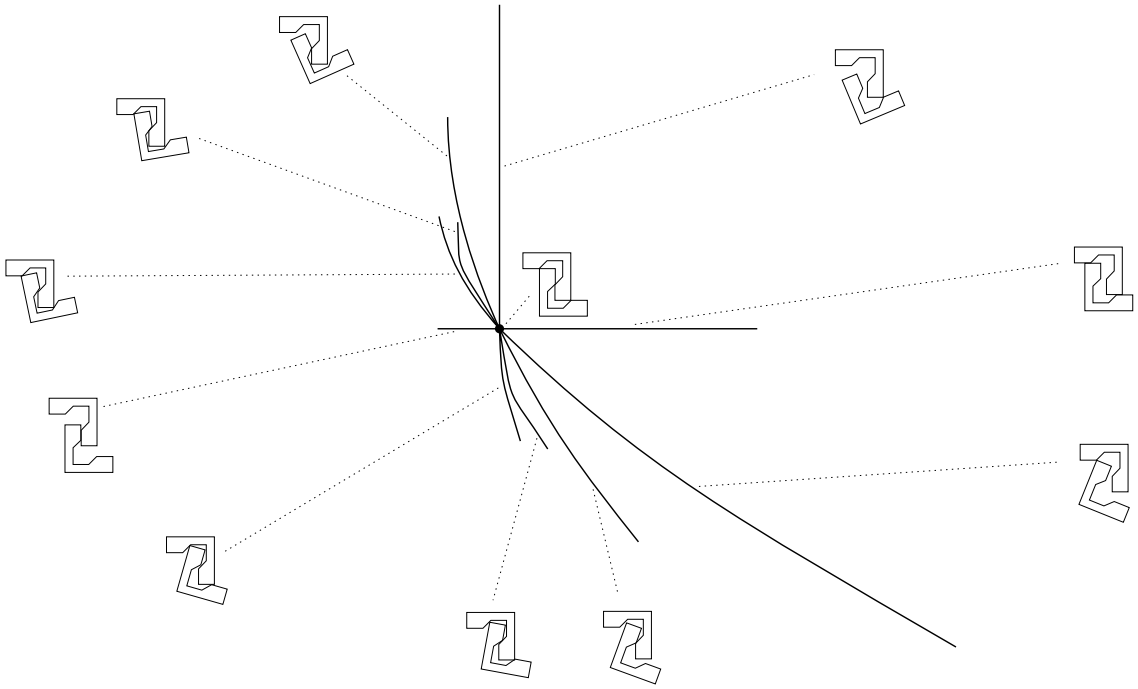
To promote robustness, several strategies were used, including explicit handling of non-generic geometric cases, the use of inherently robust numerical procedures, consistent application of fuzzy arithmetic, and thorough testing during program development. As a final test, the program was tested on a suite of 1600 examples, shown in Figure 14. Automatic correctness checks were applied to assure that certain critical invariants were never violated during program execution, and also to verify the metric and topological validity of the final output. After significant work to remove logic errors and some weaknesses in the numerical analysis, the program was able to solve all 1600 cases without an error. This result suggests that the program has good robustness, but certain key weaknesses remain; these are discussed in Part II. Whether these weaknesses may be overcome with further work remains an open question.

Several strategies were also applied to promote efficiency. These include standard optimization techniques such as caching and the construction of data structures to support constant-time execution of important operations. In addition, efficient geometric tests were developed that allowed subtantial computation to be avoided when certain convexity criteria are satisfied. Finally, the computation was organized to allow incremental construction of the obstacle surface, enabling planners to avoid constructing irrelevant obstacle features. The final program computes configuration obstacles on the order of seconds; for example, the configuration obstacle shown in Figure 6 was constructed in 18.95 seconds, while only 3.67 seconds were required to construct the four facets needed to synthesize the example pushing motion.[2] Detailed performance data are presented in Part II.

[2]All execution times reported in this thesis reflect the elapsed execution time of a Common Lisp program running on a Symbolics 3600 Lisp Machine. This machine was equipped with two megawords of memory, and did not have any special floating-point hardware.



(a)



(b)

Figure 13: Examples of difficult cases solved by the *CO* program. (a) A configuration-obstacle facet with six disconnected components. (b) A non-generic configuration-obstacle vertex with ten obstacle edges intersecting at the vertex.

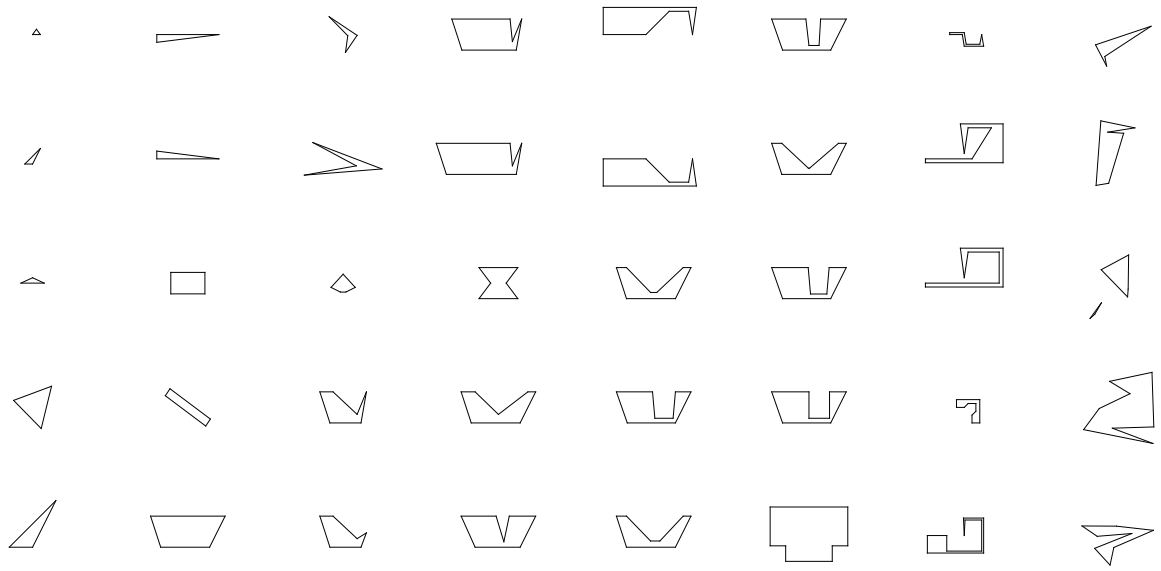


Figure 14: The test suite for the *CO* algorithm. All pairs of these 40 polygons form 1600 example problems. 35 of the polygons were hand-designed to assure that all of the possible generic and non-generic geometric cases arose in the test; the remaining 5 polygons were randomly-generated in an effort to reduce the presence of human bias.

STATIC

The first step in the analysis of the *STATIC* algorithm was to develop a representation of planar forces that allows the static equilibrium analysis problem to be reduced to the problem of deciding whether two polygons intersect. This representation was developed in collaboration with Matt Mason, and is illustrated in Figure 15. The coordinate system shown in the figure is the $(F_x, F_y, \tau/\rho)$ force-torque space. This is the space of possible planar forces; the horizontal axes describe the force direction, and the vertical axis describes the force moment. Given one or more frictional contacts, the set of possible contact reaction forces corresponds to an infinite polyhedral cone emanating from the origin [Erdmann 1984]. We project this cone onto the surface of a unit sphere centered at the origin, producing a two-dimensional representation of the possible contact reaction forces. Similarly, we express the set of possible applied forces as another polygon on the sphere; static equilibrium is only possible when these polygons intersect.

This representation was used to develop an algorithm for constructing the set of possible equilibrium states. For each feature of the configuration obstacle, the algorithm computes the subset of the feature where static equilibrium can occur. The boundaries of this subset are found by a critical-value analysis that identifies configurations where the overlap between the two polygons on the force sphere becomes singular. This critical-value analysis was carried out for all of the obstacle features that are possible, including both generic and non-generic cases. The algorithm employs a small conservative approximation when constructing the polygon of possible applied forces; this approximation can sometimes cause extra configurations to be included in the set of possible-equilibrium configurations. The approximation and its implications are discussed in Part II.

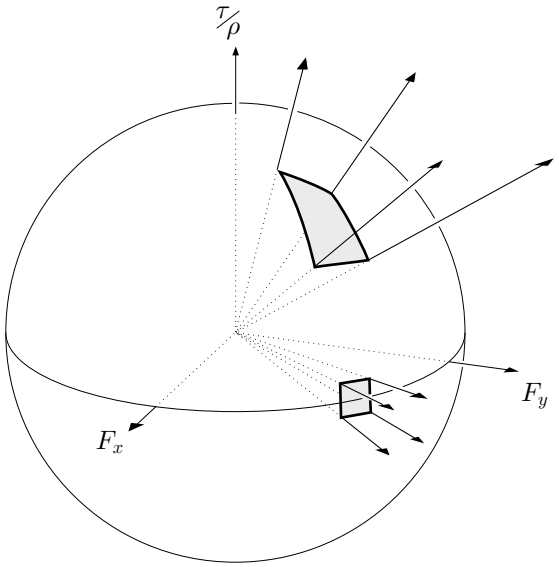


Figure 15: Projecting forces onto the surface of the unit sphere in force/torque space. Sets of forces map to polygons on the sphere; the large polygon represents the negated set of possible contact reaction forces, and the small rectangle represents the set of possible applied forces. Static equilibrium is only possible when these polygons intersect.

The algorithm was implemented, and the resulting program includes all of the cases covered by the analysis. The program was tested on a set of cases that gives rise to all of the possible critical conditions that can occur on the force sphere, for both generic and non-generic input data. Unfortunately, automatic verification of the algorithm was not performed, primarily due to the difficulty of writing suitable automatic test procedures. Instead, the correctness of the implementation was checked manually using a variety of software tools designed for this purpose. Figures 20 and 111 show examples of output produced by the *STATIC* algorithm.

Because the algorithm does not involve topological constructions, it is not prone to many of the robustness problems that made implementation of the *CO* algorithm so difficult. In the tests that were performed, no robustness weaknesses were observed; I expect that errors in the program that remain undetected will be straightforward to repair.

As with the *CO* algorithm, optimizations were included to reduce unnecessary computation. For the example pushing task described in the previous section, the program is able to identify the possible-equilibrium configurations for the entire obstacle in 30.10 seconds (see Figure 111). If the planner constructs only the four facets necessary to synthesize the example pushing motion, then the static-equilibrium analysis only requires 8.55 seconds.

BP_e , BP_i , and *COMMAND*

While the *CO* and *STATIC* algorithms have been fully developed and tested, this work is not yet complete for the BP_e , BP_i , and *COMMAND* algorithms. As a result, some of the necessary analytical details have not been worked out for these algorithms, and the resulting implementations contain numerous inefficiencies, bugs, and numerically weak procedures.

Nonetheless, these programs successfully generated the data needed for the physical experiments described below, as well as the example computations presented in this thesis. The following paragraphs briefly summarize the current status of these algorithms. Computation times are presented, but these should be viewed with skepticism; the final implementations of the algorithms will be slowed by the presence of additional code to handle unsupported geometric cases, and will be sped up by more efficient programming. Robustness results are not available.

The BP_e algorithm involves (1) identification of critical points where potential-energy constraints apply, (2) sorting the resulting points by their height in the potential energy field, (3) using the lowermost point to construct a pair of half-space constraints, and (4) constructing a connected configuration-space volume that contains the goal and obeys the half-space constraints. In the current implementation, the critical points must be selected manually, and manual constraints must sometimes be added to insure that the constructed volume is connected. Otherwise, all of these steps are performed automatically. After the necessary manual specifications were provided, the current BP_e program constructed the volume illustrated in Figure 8 in 14 seconds.

The BP_i algorithm involves construction of an initial contour surrounding the goal, expansion of this contour to produce a region on the configuration obstacle surface, and then ray-construction to lift the two-dimensional region into a three-dimensional volume. The current implementation of this algorithm is completely automatic for simple problems, but several important geometric cases are not supported. Further, the current implementation is relatively slow — for example, the pushing contour shown in Figure 10(d) required 38 minutes to construct. There are three factors that contribute to this slow execution: *(i)* the implementation is extremely inefficient, *(ii)* the trajectory function D is expensive to compute for pushing mechanics in the presence of uncertainty, and *(iii)* the current integration algorithm requires $O(n^2)$ steps, where n is the number of points in the final contour. Part II suggests ways to address each of these issues, and substantial speed improvements may be possible.

Compared to the other algorithms, the *COMMAND* algorithm is very simple. In the current implementation of this algorithm, the volumes V_{state} and V_{command} are represented by lists of polygonal slices. Shrinking a volume is then performed by shrinking each polygonal slice by ϵ_{xy}, and then intersecting each resulting slice with all of the slices within a vertical distance of $\pm\epsilon_\theta$. Implementing this algorithm is not difficult, but requires correct handling of a plethora of geometric cases; not all of these are supported in the current implementation. The current implementation automatically constructed the volume shown in Figure 11(c) in 10 seconds.

Physical Experiments

This thesis is ultimately concerned with the goal of building reliable autonomous manipulation systems. Therefore it is essential that these methods produce planners and analysis techniques that are successful in the physical world.

A theoretically sound implementation of the proposed algorithms will produce actions that will succeed in the idealized world of our physical model, and if our physical model is a sufficiently accurate representation of the real world, then these actions should succeed in the real world as well. If we then perform these actions in the real world as an experiment, successful results will support the vailidity of both the algorithm and the model, while unsuccessful results will suggest that there is a problem with the either the algorithm or the model, or both.

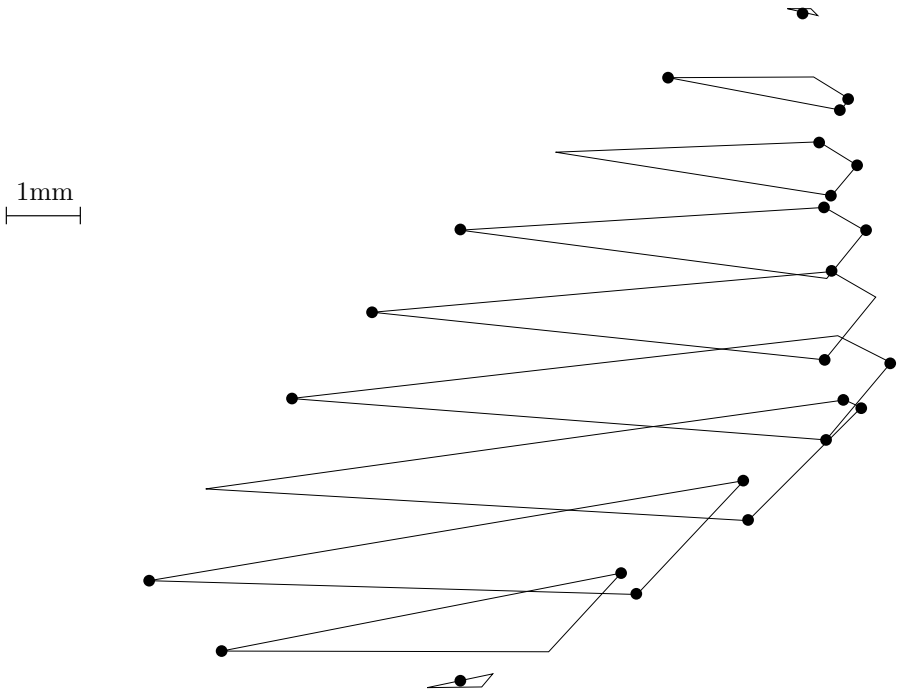


Figure 16: The 25 sample points chosen for experiment #1. The polygons are the slices of V_{command} shown in Figure 11(c), viewed from above.

This section summarizes four physical experiments designed to test the physical validity of the results produced by the programs described above. Two of these experiments involve pushing mechanics, and two of the experiments involve mechanics in the presence of a gravity field.

Experiment #1: Exploring the Worst-Case Conditions

The volume V_{command} shown in Figure 11(c) represents a set of commanded starting positions that will produce the desired final configuration, even under the worst-case combination of uncertainty conditions. The goal of this experiment is to test that assertion.

An ideal test would be to consider every point in V_{command}, and then for each such point, execute a pushing operation under all possible combinations of uncertainty conditions consistent with the input uncertainty parameters. If the program's analysis is correct, then the goal configuration should be achieved in every trial.

Of course, this ideal experiment is impossible, because it requires an infinite number of trials. A more practical experiment would be to choose a finite number of points in the volume, and test each point under a finite number of uncertainty conditions. These points and uncertainty conditions could be randomly chosen, but a better strategy is to intentionally choose points on the boundary of the volume, and apply the maximum amount of uncertainty to make each trial as difficult as possible.

Using this strategy, I selected 25 points distributed over the boundary of V_{command}, and used an industrial manipulator to execute 48 pushing operations for each point. These points are shown in Figure 16. In each operation, I added a controlled amount of uncertainty, within the limits specified

as input to the algorithm. The 48 pushing operations executed for each point corresponded to all combinations of extremal uncertainty conditions in five variables: support friction distribution, object position, object orientation, robot orientation, and robot pushing direction. The magnitude of some of these errors was reduced slightly from the input value to compensate for the presence of natural uncertainty in the experimental apparatus. The remaining input uncertainty parameters were the object center of mass location, coefficient of friction, and robot position. These were not controlled in the experiment, because the input uncertainty parameters closely matched the true uncertainty of the physical apparatus.

These choices produced a set of 1200 experimental trials. The trials were executed using an interactive computer program; I would set up the task initial conditions, tell the program to execute the trial, observe the result, and manually enter whether the trial was a success or failure. These results were then automatically tabulated. A video camera was used to record each experiment so that I could re-examine doubtful observations using instant replay.

All 1200 trials successfully achieved the goal. It should be noted, however, that I observed an unexpected phenomenon during the experiment. During the execution of some of the pushing motions, the object was observed to "chatter" after achieving the final configuration. This chattering was probably due to lack of smoothness in the robot's motion, or possibly arose due to a combination of stick/slip and deformation effects in the interaction with the support surface. The chattering seemed to diminish when the motion speed was reduced, and I have not observed it while executing a pushing motion by hand. Regardless of the cause, this was not a behavior that was predicted by the planner. Note that the goal configuration remained stable even in the presence of this unmodelled effect; I conjecture that this is due to the inherent stability of the task geometry, as predicted by the energy analysis displayed in Figure 8.

Experiment #2: Including Constraints

The algorithms described above are defined to include the presence of task constraints; these are states that should never occur during task execution. These constraints are easy to include in the BP_e and BP_i algorithms, because the constraint states may be treated as non-goal possible-equilibrium states. The resulting commanded actions should succeed in achieving the goal, without violating the task constraints. This experiment is designed to test this assertion.

To test the planner's ability to include constraints, we will consider an example task with and without constraints. We will execute the resulting plans, and insure that the plan constructed with constraints never violates the constraints during plan execution. It may also be interesting to observe the plan constructed without constraints; if the executed motion violates the constraints, then we know that the planner included an explicit compensation to avoid the constraint states.

The example task is shown in Figure 17. The final configuration is shown, and we will assume that the goal of the task is to glue the objects together in this configuration. Further, we will assume that some previous operation has applied contact cement to the large polygon on the edge indicated in the figure. Because this cement instantly bonds to anything it touches, it is important that nothing touch the edge until the desired configuration is achieved.

Figure 18 shows a plan that would be constructed without the contact cement constraints. As with the previous pushing example, the pushing direction was selected by hand. For this pushing direction, the volume of initial states that will achieve the desired configuration is very large —

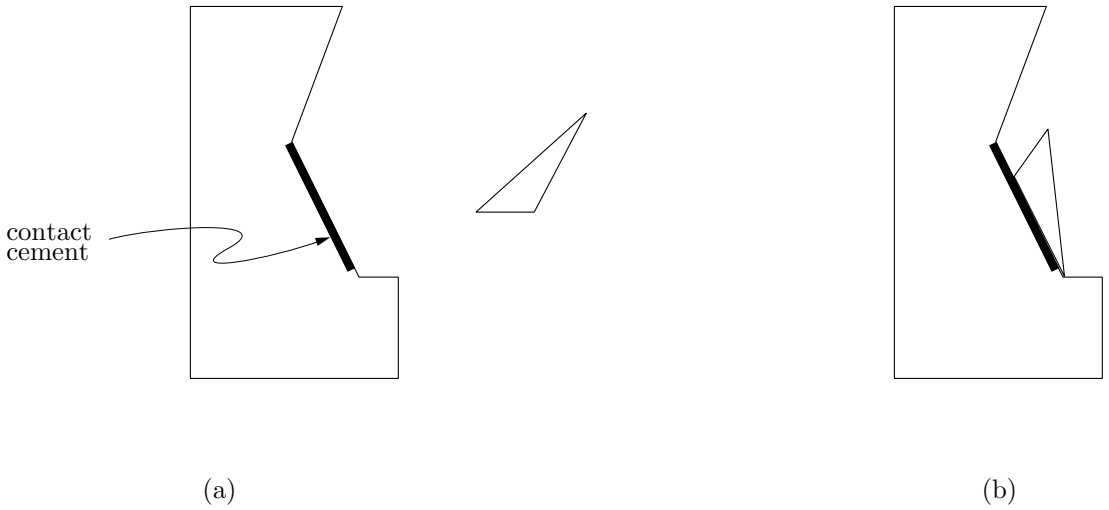
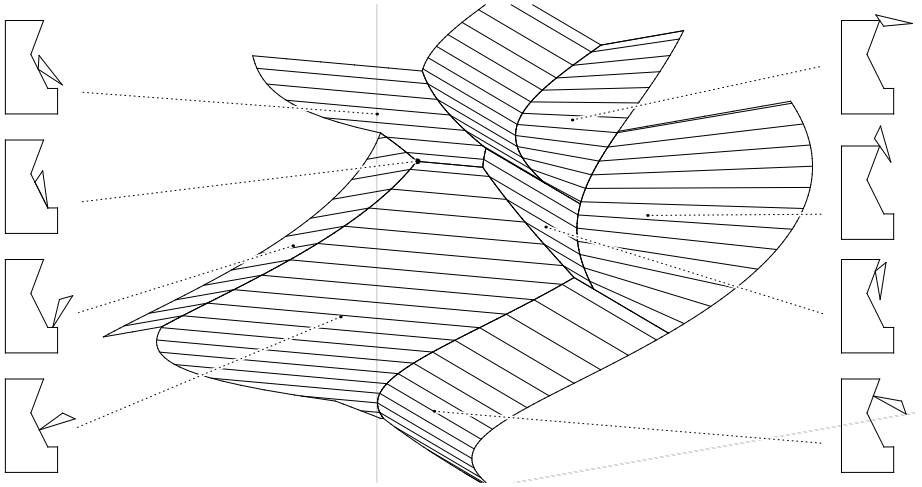


Figure 17: A task involving constraints. (a) The objects. (b) The goal configuration. This task is complicated by the constraint that the contact cement should not touch anything until the final configuration is achieved.

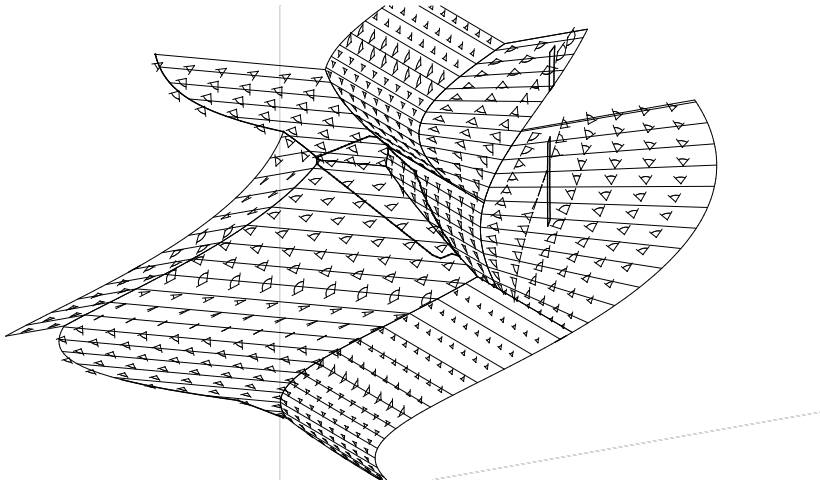
larger than the volume shown, in fact. In this case the integration process was terminated early because the constructed volume was already large enough to compensate for the task uncertainties, and because of limitations in the implementation. A nominal point was chosen from the resulting volume, and the corresponding pushing operation was executed manually. A custom-made jig was used to insure that the executed motion was a true linear motion with the desired motion direction and finger orientation. Figure 18(d) depicts the observed motion; notice that this motion would be unsuitable if the contact cement was present, since the object slides down the cement-covered edge toward the goal.

Figure 19 shows a plan constructed including the contact cement constraints. The constraints were expressed by identifying the configuration obstacle features corresponding to contacts with the cement-covered edge, and marking the subset of those features where a contact occurs with the cement. The resulting obstacle regions are highlighted in the figure. The BP_i program was then run normally, given these regions as the task constraints C, and a different manually-selected pushing direction. The resulting volume of initial states is shown in the figure. Once again, a nominal point was chosen from the volume, and the corresponding pushing motion was manually executed. Figure 19(e) depicts the observed motion; in this case, the constraints were obeyed consistently over the execution of several trials.

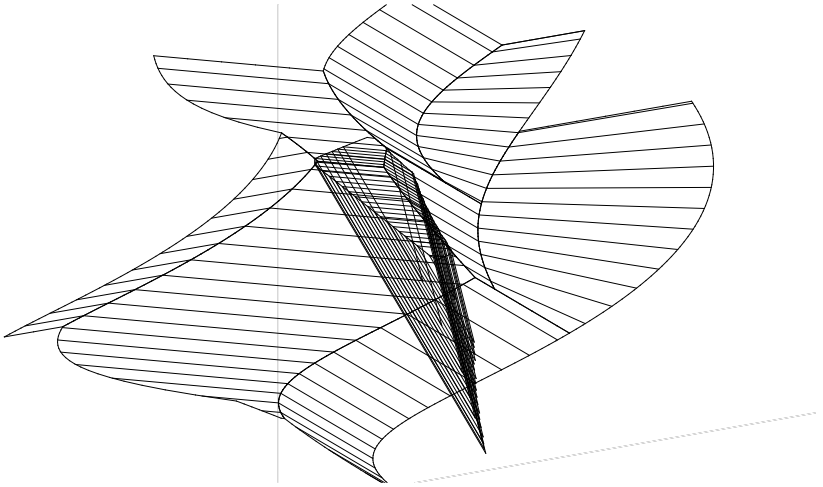
This experiment demonstrates the ability of the approach to include the presence of task constraints. However, the current implementation requires a manually-selected pushing direction; will a fully automatic planner be able to replicate these results? This remains an open question, but I conjecture that the answer is yes, for the following reasons: First, unsuitable pushing directions can be automatically rejected, because the resulting volume of commanded starting positions will be null. Second, there is a finite interval of acceptable commanded pushing directions for each task; a generate-and-test method of selecting pushing directions will find a member of this interval, as long as the search granularity is sufficiently fine. Finally, it is possible to exploit local geometric features of the task to guide the search for a suitable pushing direction. Future work will investigate these issues.



(a)

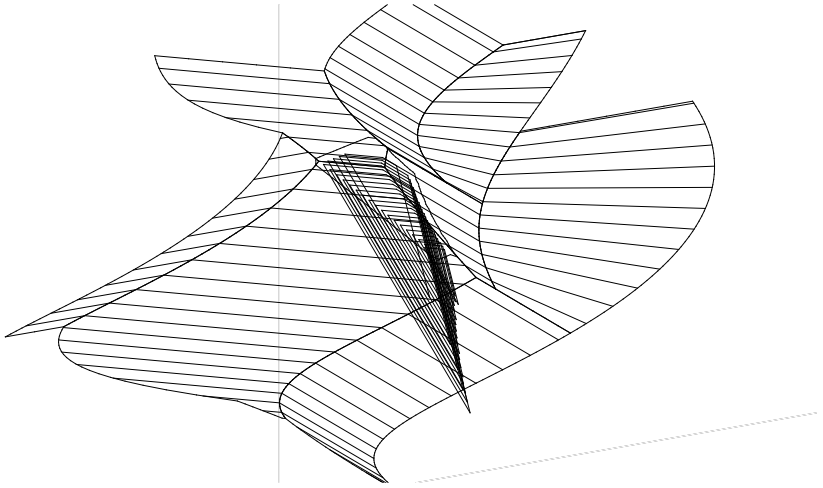


(b)

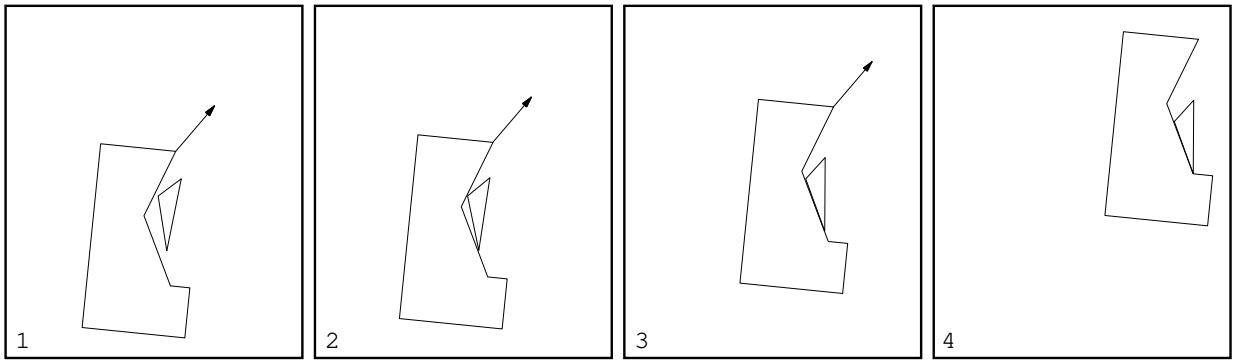


(c)

Figure 18: Constructing a volume of initial states that will converge to the goal, not including the contact-cement constraint. (a) The goal and relevant configuration-obstacle features. (b) The result of the BP_i contour integration. (c) The volume V_{state}.



(d)



(e)

Figure 18 (continued): (d) The volume V_{state} after shrinking for object position uncertainty. V_{command} was also constructed, but is not shown. (e) A typical pushing action selected from the volume; this action corresponds to a point near the center of the volume. Here $\psi = 55° \pm 2°$.

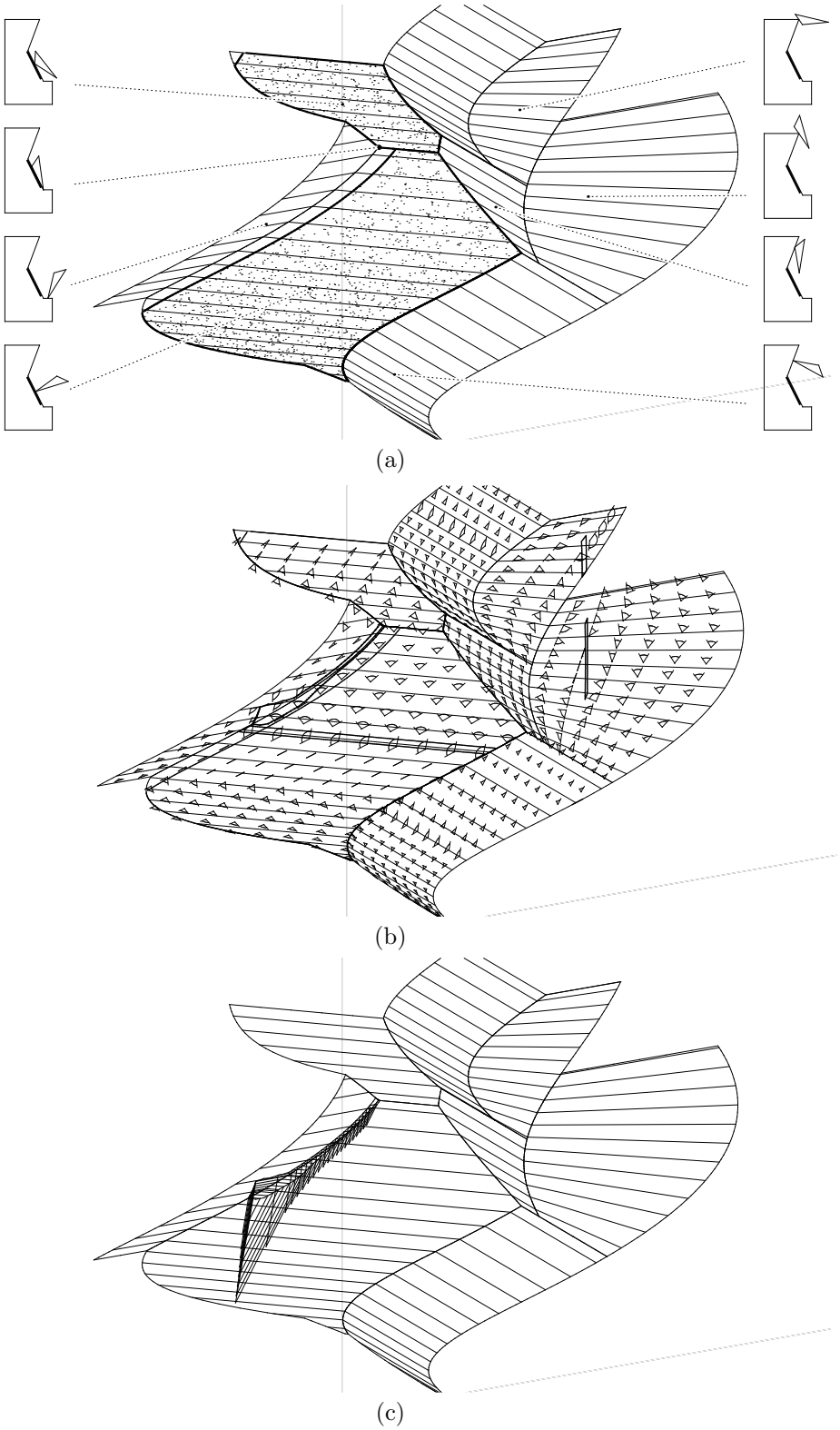
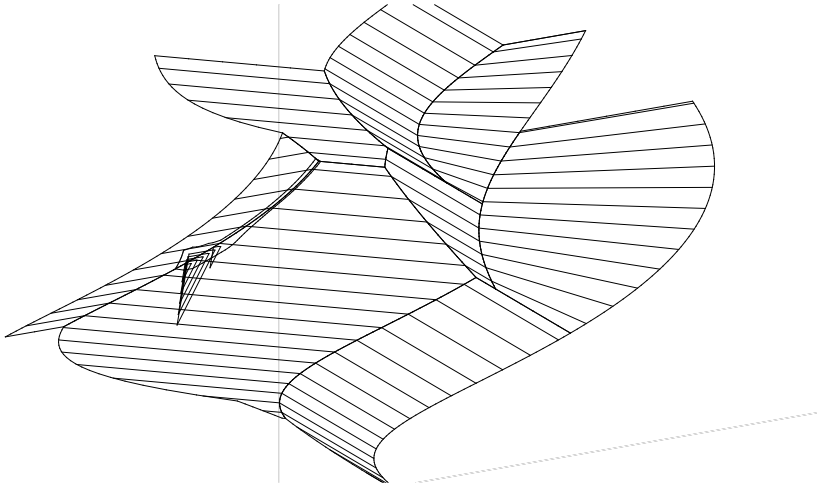
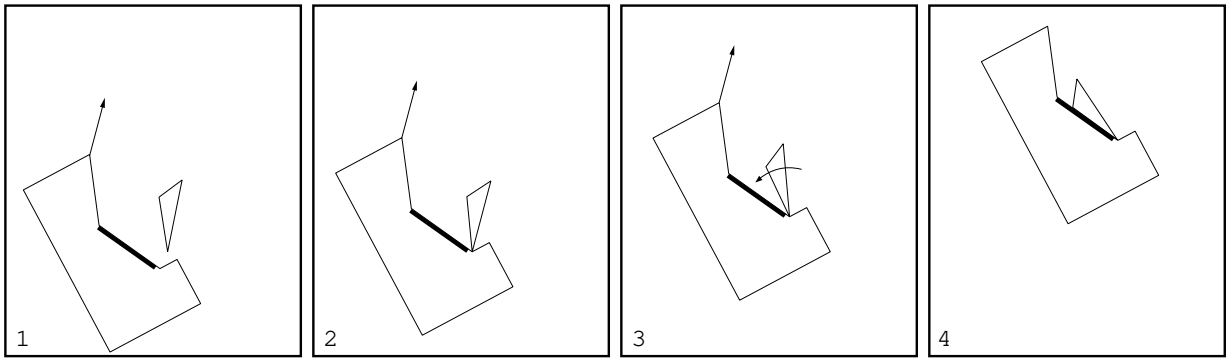


Figure 19: Constructing a volume of initial states that will converge to the goal, obeying the contact-cement constraint. (a) Expressing the constraints. The shaded regions on the obstacle surface are configurations where the constraint is violated. (b) The result of the BP_i contour integration. (c) The volume V_{state}.



(d)



(e)

Figure 19 (continued): (d) The volume V_{state} after shrinking for object position uncertainty. (e) A typical pushing action selected from the volume; this action corresponds to a point near the center of the volume. Notice that the contact cement is not touched until the final configuration is attained. Here $\psi = 46^\circ \pm 2^\circ$.

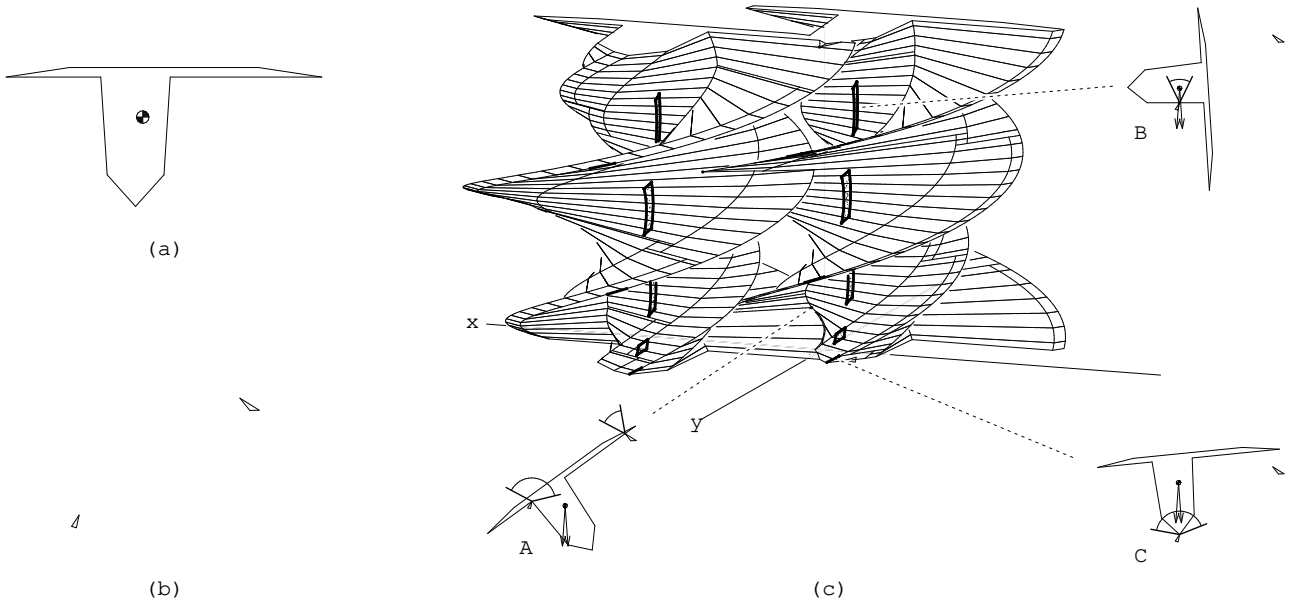


Figure 20: A part and an orienting fixture. (a) The part. (b) The fixture. (c) The configuration-space obstacle of the part and fixture, with the possible-equilibrium states identified. The configuration labelled A is the only stable configuration; B and C are examples of unstable equilibrium configurations.

Experiment #3: Parts Nest Analysis

The *STATIC* algorithm computes the set of all configurations where static equilibrium is possible. Conversely, the complement of this set is the set of states where motion *must* occur. The goal of this experiment is to test this assertion, and to explore the usefulness of the *STATIC* algorithm for the analysis and design of part-orienting fixtures.

As mentioned above, the *STATIC* algorithm may be used to identify all of the possible rest configurations for a part dropped into a fixture. This output can then be used to verify that equilibrium is possible for the goal configuration, and also to identify the undesirable rest configurations that must be removed with mechanical filters. In some cases, it may be possible to design a fixture where the desired configuration is the only stable rest configuration.

Figure 20 shows a part and fixture with only one stable rest configuration. The configuration obstacle is also shown in the figure, with the possible-equilibrium states identified. There are many possible-equilibrium states, but only one of these states will remain stable in the presence of small perturbations. The implies that if the part is dropped onto the fixture, it will either come to rest in a known stable configuration, or it will fall through to be dropped again.

The design of this part and fixture was facilitated by the *STATIC* program. I began with an initial part design, and an idea for an orienting fixture. I then ran the *STATIC* program on these designs, and inspected the resulting stable rest states. Seeing the stable rest configurations and their associated force-sphere situations suggested modifications to the fixture or part to remove the undesired rest state. After a few iterations of this process, I arrived at the final design in less than two hours. In so doing, I made several modifications to the fixture, and a small modification to the part.

I then fabricated the part and fixture to test the physical validity of the program's predictions. I attached the fixture to a flat support surface, and elevated this surface to a large angle. I then performed dropping experiments by placing the part on the surface above the fixture, and releasing it. The part would then slide down the surface onto the fixture, experience one or more collisions, and then either come to rest on the fixture or continue to fall. Over the course of several trials, the only rest state I observed was the predicted stable configuration.

I also tried to carefully place the part into other stable configurations. In some cases I was able to obtain unstable equilibrium conditions like the configuration labelled C in Figure 20, but these configurations did not persist in the presence of a small perturbation (a small discrete impact applied to the support surface). These observations are consistent with the predictions of the *STATIC* program. However, there was a surprise: Configurations similar to the configuration labelled B in Figure 20 did remain stable, sometimes even after several small perturbations were applied.

This unpredicted behavior arose because of the presence of support friction between the part and the supporting surface. This support friction was not included in the *STATIC* algorithm's model of this physical system, which explains why it was not predicted by the program. To resolve this problem, we can either extend the *STATIC* algorithm's analysis to include this unmodelled phenomenon, or we can modify the world so that the phenomenon is no longer present. I chose the latter, and added a vibration source to the support surface. This made the support friction effectively nonexistent, and the unpredicted behavior was no longer observed.

Experiment #4: Placing by Dropping

The *STATIC* algorithm constructs the set of states where equilibrium *may* occur; if we want to know that equilibrium *will* occur in a given task, more computation is required. For dropping tasks, the BP_e algorithm may be used to construct a set of initial dropping positions that will reliably achieve a desired final configuration. The goal of this experiment is to test this assertion.

The BP_e program was applied to the part and fixture defined in the previous experiment, with the previously-identified stable configuration specified as the goal. This program calls the *STATIC* algorithm to identify possible equilibrium states, and then performs a subsequent energy-constraint analysis. The resulting volume of initial dropping positions is shown in Figure 21. This volume corresponds to a set of configurations where the part center of mass is below the maximum-height line shown in the figure, the wide base of the part is between the fixture polygons, and the two arms of the part are above the fixture polygons. Figure 22 shows some example configurations inside the volume.

One possible experiment to test the physical validity of this result would be to systematically explore initial dropping positions within the volume, under controlled uncertainty conditions. Such an experiment would be technically difficult, because it requires a method of placing the part at a precisely known initial dropping position, and then releasing the part instantaneously without interfering with its motion. As with the first pushing experiment, we would like this method to be as automatic as possible, so that a large number of trials may be executed.

Sufficient time was not available to devise and implement a suitable automatic apparatus, so instead I performed a less rigorous manual experiment. I carefully measured and drew the maximum-height line on the surface containing the fixture polygons, and similarly marked the location of the

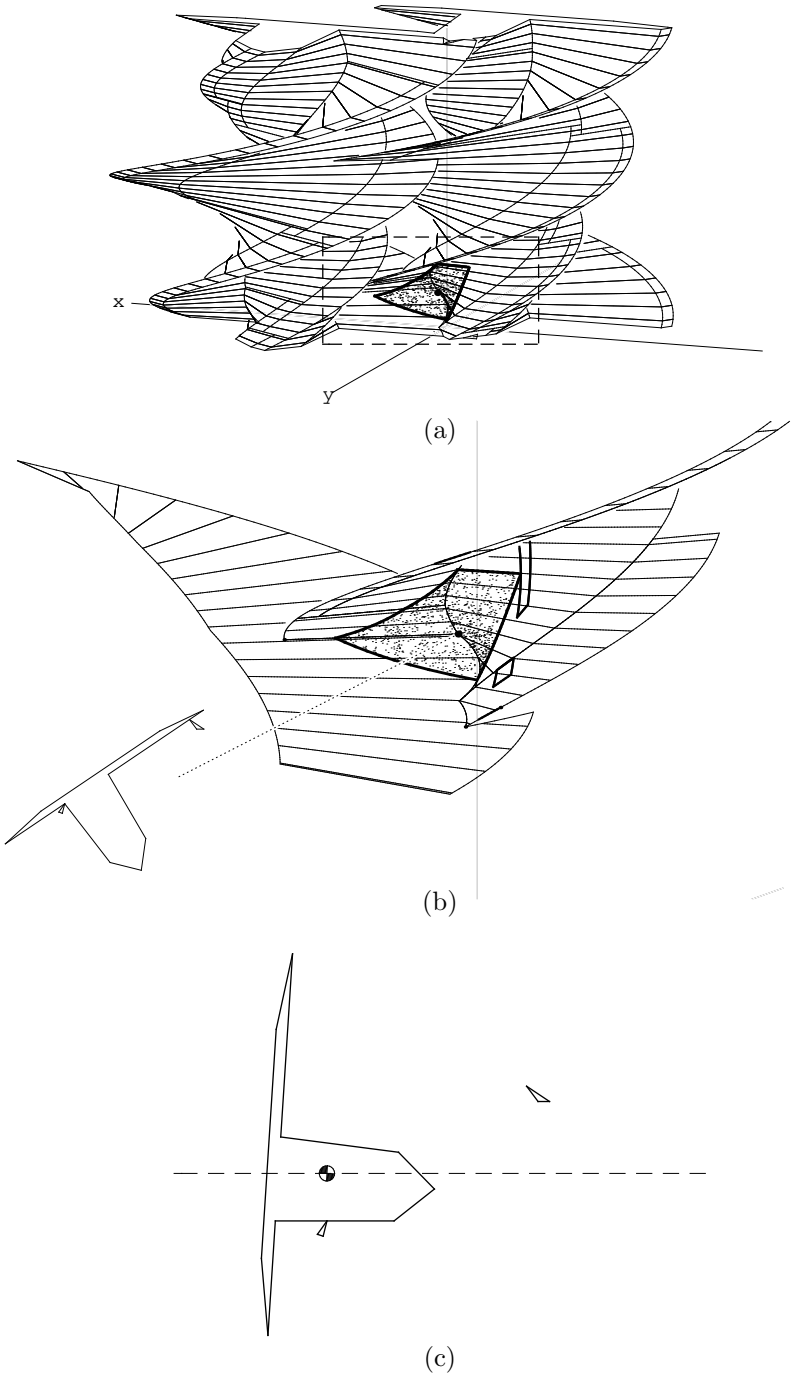


Figure 21: Analysis of a placing-by-dropping operation. (a) A volume of initial dropping positions that will reliably achieve the goal configuration. This volume is a "puddle" filling a concavity in the configuration-space obstacle; the shaded region is the bottom of the puddle. (b) An enlarged view of the volume. (c) The critical configuration that gives rise to the maximum-height constraint. Initial dropping positions higher than this configuration have sufficient energy to allow the object to escape while translating left in the displayed orientation. If the initial dropping height is lower than the height in this configuration, then the object cannot escape. This configuration corresponds to the upper-right corner of the volume; if more water is poured into the "puddle," then it will leak past this point. The volume includes ± 4 degrees of gravity direction uncertainty; this has been omitted in (c) for clarity.

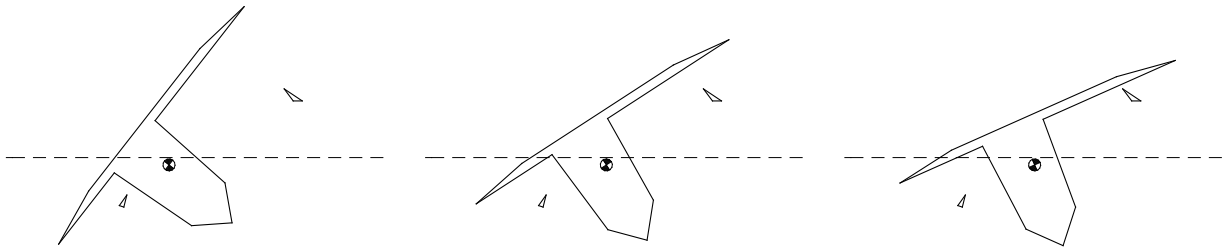


Figure 22: Typical configurations within the volume shown in Figure 21.

part center of mass. I then executed a number of dropping operations by releasing the part from initial configurations that obeyed the maximum-height and other constraints. In all trials, the part settled into the predicted configuration, even after bouncing multiple times before settling. I also executed dropping actions from initial positions higher than the maximum-height line, and found a large additional region of initial dropping positions that reliably produced successful results. Executing these operations gave me the impression that the volume constructed by the BP_e program was so reliable that the rigorous experiment described above seems uninteresting.

Is the Worst-Case Analysis Too Conservative?

The programs described in this thesis are designed to compute a set of commanded actions that will reliably achieve the goal. This computed set is a conservative approximation of the true set of actions that will reliably achieve the goal; there will generally be points outside the computed set that will also reliably succeed. An important question is: How closely does the computed set approximate the true set? If the true set of successful actions is much larger than the computed set, then the algorithms may fail to find solutions to problems that are not actually difficult.

The BP_e algorithm intentionally uses a weak model of the task mechanics, so we would expect the sets constructed by this algorithm to be much smaller than the corresponding true sets of reliable actions. The observations in Experiment #4 support this expectation, since a region of release positions above the maximum-height line was found to reliably achieve the goal. These observations emphasize the advantages and disadvantages of using a weak mechanics model: We may avoid a substantial amount of analysis and computation, but at the same time risk missing solutions in cases where reliable actions are available.

The BP_i algorithm uses a strong model of the task mechanics, so we would expect this algorithm to compute a set that closely approximates the true set of reliable actions. It should be the case that every point in the computed set succeeds under all combinations of uncertainty conditions, while points not-too-far outside the set may fail for some choices of uncertainty conditions. Observing the 1200 successful pushing actions in experiment #1 led me to doubt this assertion, because the actions appeared to be so robust that it seemed that the true set of successful actions must be substantially larger than the computed set.

To test this doubt, I conducted an experiment designed to explore the nature of points outside the computed set. In this experiment, I selected three points outside the volume of commanded starting positions V_{command}, and also identified the nearest points on the boundary of the volume. These points are shown in Figure 23. I then executed the corresponding pushing actions, with no uncertainty added; all of these trials succeeded. Next I examined each of the points outside the

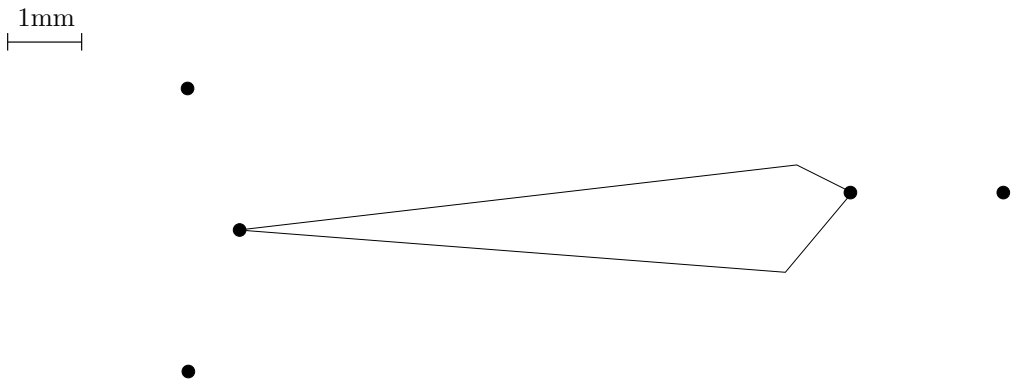


Figure 23: Sample points chosen outside the volume V_{command}. The polygon is the middle slice of the volume shown in Figure 11(c); the chosen points are 2mm outside this polygon. The nearest points on the boundary of V_{command} are also shown.

volume, and attempted to characterize the worst-case uncertainty condition consistent with the input uncertainty parameters. I then executed the pushing operations again, explicitly including the worst-case uncertainty plus an additional small margin of uncertainty to compensate for the natural uncertainty of the apparatus. This time, the points outside the volume failed, while the neighboring points on the volume boundary succeeded under the same uncertainty conditions. Figure 24 shows the failure modes that were observed.

The results of this experiment suggest that my doubts were misguided, and that the computed volume V_{command} does represent a close approximation to the true set of reliable pushing actions for this task. This further suggests that if we are interested in planning actions that are 100% reliable, the BP_i and *COMMAND* programs are not overly conservative. However, there is more to be said about this topic, and we will return to it in the next section.

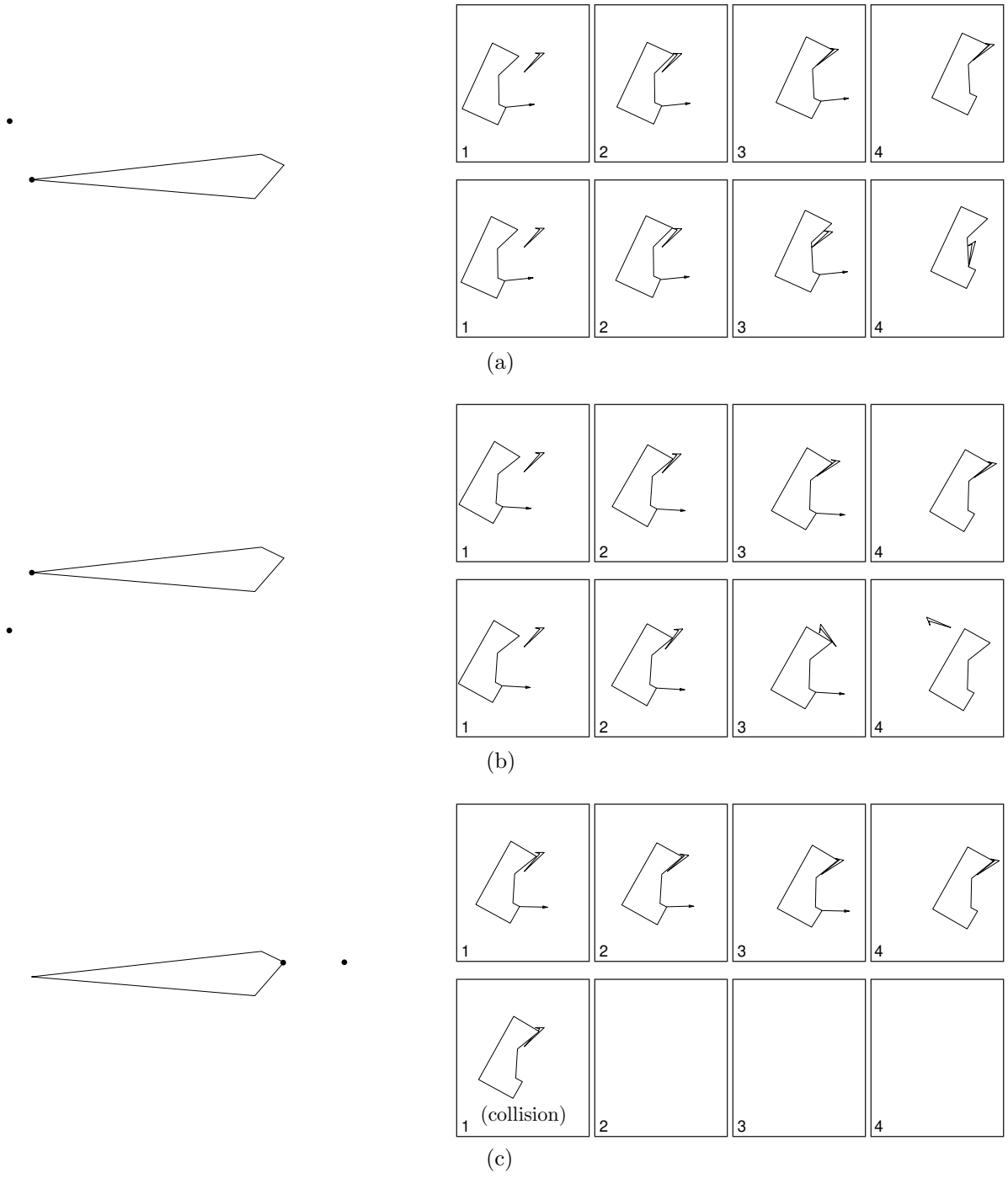


Figure 24: Observed failure modes for points outside the volume V_{command}. (a), (b), and (c) show the failure mode for the three points chosen outside the volume. In each case, the figure shows the chosen point and volume slice, along with two motion traces. The upper trace shows the observed motion for the point on the boundary of V_{command}, and the lower trace shows the observed motion for the point outside V_{command}. The uncertainty conditions are the same in both traces, but differ across (a), (b), and (c). The failures shown in (a) and (c) occurred consistently, while the failure shown in (b) occurred roughly half the time, due to the presence of natural uncertainty. These observations are consistent with the program's predictions, because points in V_{command} reliably succeed, while points outside V_{command} possibly fail.

What Have We Learned?

This thesis began by describing an important problem facing robotics research: The problem of automatically constructing reliable actions that move objects into desired positions. This problem is difficult because of the presence of uncertainty, which inherently exists in any real manipulation task. Further, correct solutions must obey the constraints imposed by the task geometry and mechanics, as well as other constraints that might be imposed by the goals of the task.

The thesis proposed an approach to solving a class of manipulation problems that involve planar rigid objects. The approach exploits the continuity present in these tasks to model sets of actions as geometric sets in various coordinate spaces. Uncertainty is explicitly included by assuming that every physical parameter except shape can be represented by a maximum error value, which bounds the discrepancy that can exist between the robot's model of the parameter and the true physical parameter. The task geometry and mechanics are then used to construct a set of actions that will succeed, despite the worst-case combination of these errors. The thesis expresses this approach by defining a collection of five generic algorithms that may be applied to a variety of manipulation problems, provided that the necessary assumptions are met. These algorithms are then applied to three example problems: synthesis of linear pushing actions, analysis of parts-orienting fixtures, and synthesis of placing-by-dropping actions.

The thesis then presented an experiment designed to investigate the viability of the proposed approach. The experimental procedure was to develop and implement the proposed algorithms, and then measure the performance of the resulting programs. Performance metrics included robustness, execution speed, and physical validity of the program output. This experiment is not complete, but the results so far are encouraging: The programs have moderate to good robustness, run in reasonable amounts of time,[3] and produce accurate predictions of the physical systems that were tested. Further, the programs demonstrated the ability to accomodate task constraints, and construct plans that were not excessively conservative.

[3]Assuming that the performance of the BP_i program will improve once it is completely implemented.

In developing the algorithms, the thesis produced several key conceptual results:

- A complete kinematic analysis of the interaction of a pair of polygons confined to a plane. This analysis identified all of the possible configuration obstacle features that can arise for an arbitrary pair of interacting polygons, and developed analytic equations to describe the metric properties of each feature. The analysis includes non-generic obstacle features, which correspond to degenerate multiple-contact situations which must be identified for correct physical analysis.

- A representation of planar forces, expressed as polygons on the surface of a sphere in force space. This representation simplifies the analysis of multiple-contact friction problems, and facilitates static and dynamic analysis of planar physical systems in the presence of uncertainty in contact friction and applied force. This representation was developed jointly with Matt Mason.

- A method for combining energy constraints with the configuration-space obstacle geometry to identify a set of initial configurations that will reliably converge to a desired final configuration, without requiring a model of the mechanical system trajectory. This method is especially useful for predicting the behavior of mechanical systems that involve complex collision mechanics in the presence of uncertainty.

- A representation of the behavior of a planar mechanical system, expressed as a field of non-deterministic state-transition cones embedded in the configuration space. These cones are defined for all points on the configuration-space obstacle surface, as well as in the free space surrounding the obstacle. This field completely describes the set of possible motions for all possible configurations of the system, and can include the effects of all sources of uncertainty that affect the physical system behavior. This representation facilitates subsequent analysis to predict the set of configurations that may possibly follow a given initial configuration, or to construct a set of initial configurations that will reliably converge to a desired final configuration.

Combined with the *CO, STATIC,* BP_e, and BP_i algorithms, these conceptual results comprise the primary scientific contributions of this thesis. These contributions significantly extend our ability to analyze planar mechanical systems, and to synthesize solutions to planar manipulation tasks.

Much work remains. The robustness of the *CO* program should be further improved, the analysis and implementation of the BP_e and BP_i algorithms needs to be completed, techniques must be developed to automatically identify a successful pushing direction and distance, and more experimental verification should be performed. In resolving these issues, some difficulty may arise that severely impacts the practical utility of these results — time will tell.

But suppose that these issues can all be successfully resolved. What then? What do these results imply for future work in the analysis and planning of manipulation tasks? This section explores this question. We begin by examining extensions to the current work that appear feasible, and then proceed to discuss more ambitious extensions that may prove to be very difficult. Finally, we conclude the section by discussing some important manipulation problems that appear to fall outside the scope of this approach, requiring a completely different solution.

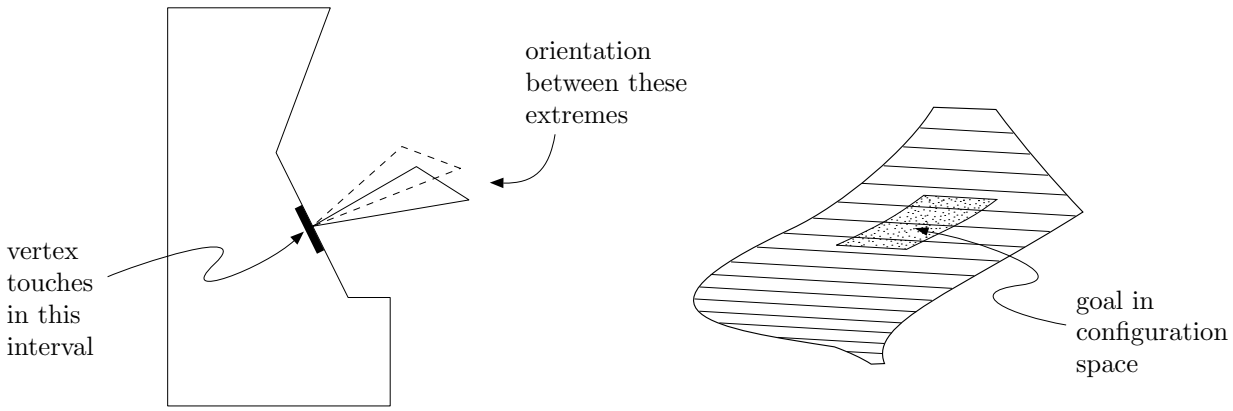


Figure 25: A goal that requires a sense of time. To synthesize a pushing action to achieve this goal, a minimum and maximum pushing distance must be computed.

Extensions

Other tasks. This thesis shows how the generic analysis algorithms may be used to synthesize pushing motions, analyze part fixtures, and synthesize dropping motions. The results of the thesis may be applied to other tasks as well.

The BP_i algorithm may be applied to any task where the possible differential motions may be computed using configuration information alone; this criterion is satisfied by several control laws, including compliant motion using generalized spring or generalized damper control, and some control schemes combining tactile and visual feedback. Single-step plans for actions under these control laws may be synthesized by constructing the appropriate differential motion function D, and then running the BP_i program.

The *STATIC* and BP_e algorithms may also be applied to other tasks, as long as the tasks meet the necessary assumptions describing the applied force. Further, it may be possible to include damping information to increase the height of the volume constructed by the BP_e algorithm. For example, in dropping tasks the coefficient of restitution may be used to bound the energy of the falling body after it experiences its first collision; coefficients of restitution significantly less than 1.0 will allow dropping heights much higher than the conservative height constructed in the examples above. The presence of ambient viscous damping (such as in underwater applications) may also be exploited to increase the dropping height.

Adding a sense of time. The algorithms presented in this thesis rely on the assumption that the goal configuration will persist indefinitely once it is achieved. While the algorithms do provide a mechanism for checking this assumption, they are incapable of constructing plans to attain goals that do not persist indefinitely, such as the example shown in Figure 25. To accomplish these goals, the planner must compute a minimum and maximum execution duration to assure that the goal is achieved; this problem is analogous to the familiar dead reckoning navigation problem. Perhaps it will be possible to extend the BP_i algorithm to synthesize duration bounds by adding extra bookkeeping to the integration process.

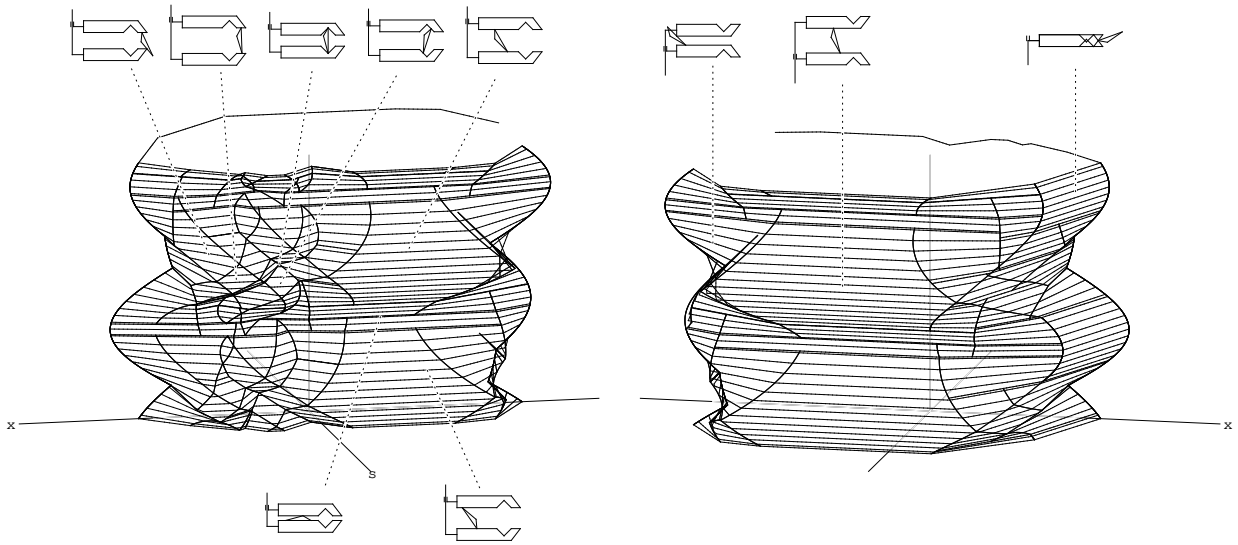


Figure 26: The configuration-space obstacle for a gripper with two fingers. The configuration space of the gripper and object has four dimensions: x, y, θ, and s, the finger separation. However, if both fingers are in contact with the object, only two degrees of freedom remain. Thus the set of configurations corresponding to two-finger contacts is a two-dimensional surface embedded in this space. This surface is displayed here in the three-dimensional (x, s, θ) space; the missing variable y is implicit.

Grasping with two fingers. A key assumption underlying the algorithms in this thesis is that there are only two objects in the world. Under this assumption, the set of possible configurations may be represented by a three-dimensional (x, y, θ) space. It is possible to extend this representation to include grippers with two fingers connected by a prismatic linkage; this extension is described in Appendix 1. The resulting configuration-space is a four-dimensional (x, y, θ, s) space, where s is the separation between the fingers. Figure 26 shows the configuration obstacle corresponding to two-finger contacts for an example gripper. By combining this two-finger obstacle with the normal configuration obstacles for each finger, it may be possible to extend the planning algorithms to construct reliable grasping actions. In addition, it may also be possible to include grippers with revolute and pantograph linkages by applying a suitable coordinate transformation when the two-finger configuration obstacle is constructed.

Challenging Extensions

Expanding the notion of plans. This thesis addresses the problem of constructing reliable manipulation plans. The approach taken thus far is to identify a set of actions that will definitely achieve the goal, even for the worst-case combination of uncertainty conditions. This condition is very strong, and results in provably robust plans. Unfortunately, in some situations the constructed set of reliable actions is null, and the current algorithms provide no information.

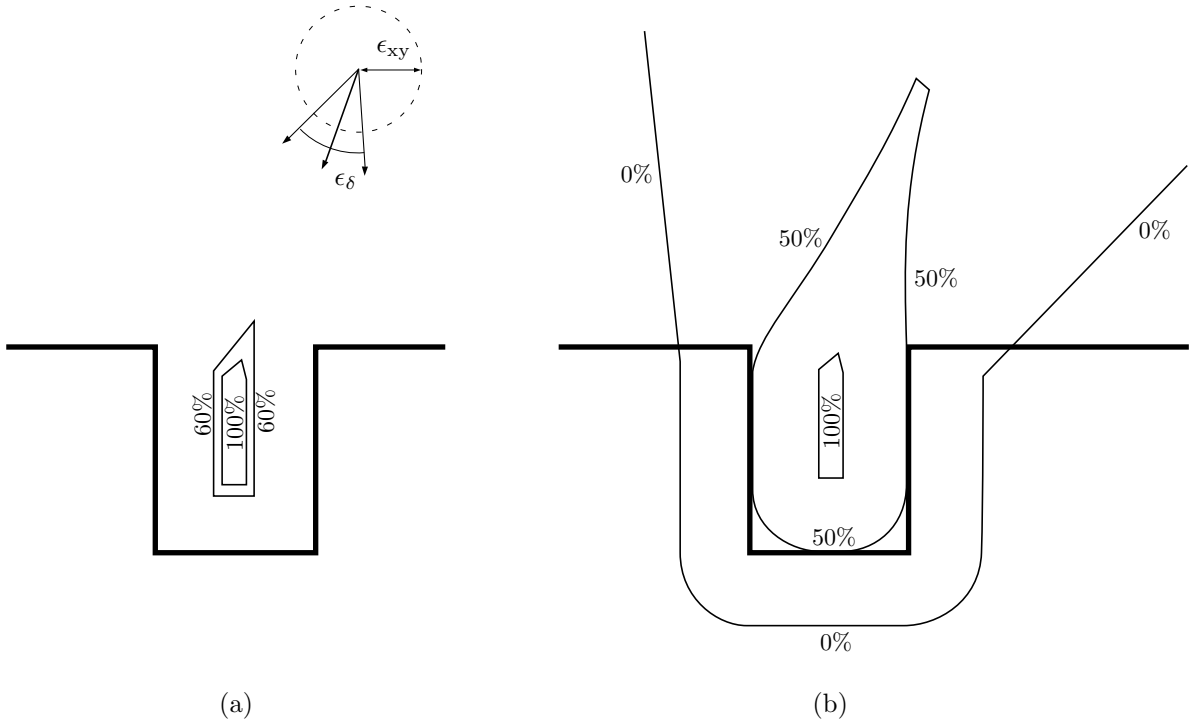
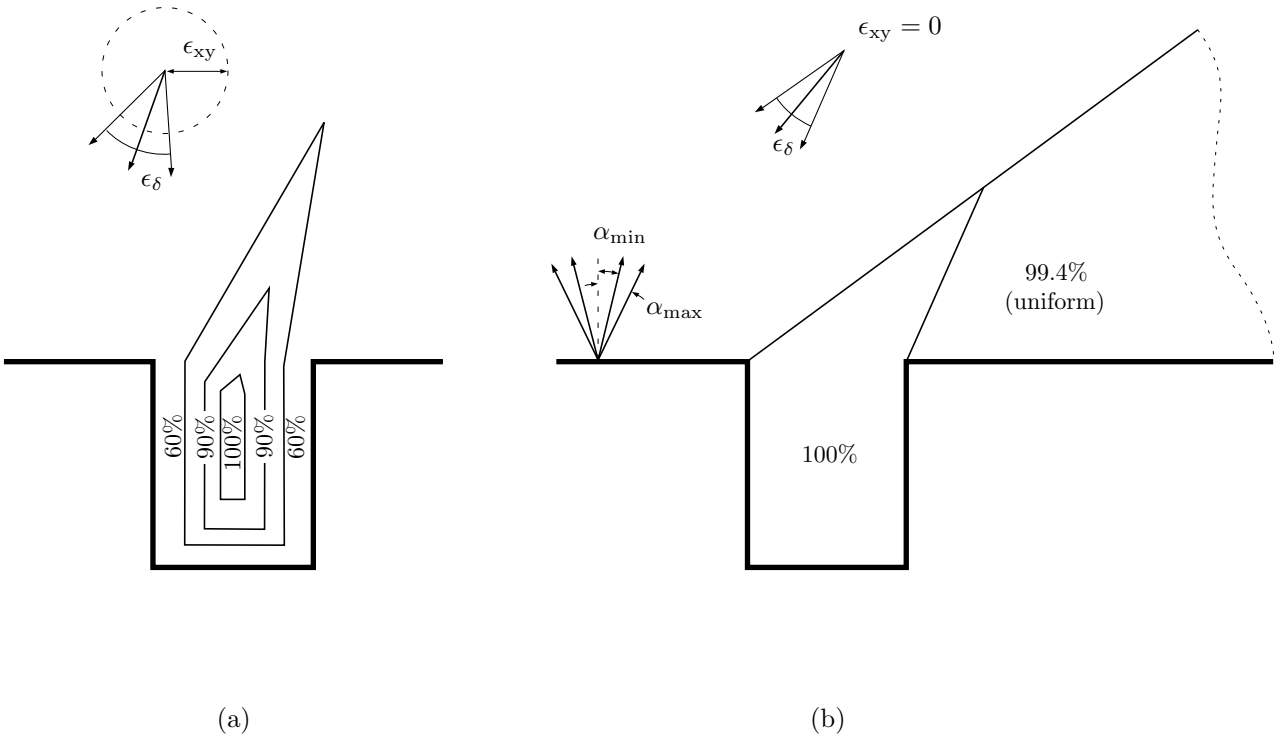


Figure 27: Computing probabilistic plans for the peg-in-hole task. (a) A set of probabilistic plans computed by running the strong planning algorithm with a reduced estimate of the task uncertainty, and then computing the probability that the true uncertainty lies within the reduced estimate. (b) The maximal probabilistic plans for the same task, constructed by manual analysis. Both diagrams assume that errors are uniformly distributed.

We can take several approaches to improving this situation. First, we could take advantage of previous results developed in the context of compliant-motion planning to extend the class of plans that may be constructed. For example, recursive backchaining could be used to construct multi-step plans that are more robust than any single-step plan [Lozano-Pérez *et al.* 1984]. Another alternative would be to construct error-detection and recovery plans that either achieve the goal or recognizably fail, allowing the task to be attempted again [Donald 1987a]. These previous results are discussed in the next section.

We could also relax our insistence on 100% reliability. For example, suppose we are given a task where 90% reliability is satisfactory. Can we compute plans for this reliability level? One approach is to use the planning algorithms we already have, but with reduced uncertainty values. The amount of the reduction is then used to estimate the success probability for the set of resulting plans.

Figure 27(a) shows the result of applying this approach to the classic peg-in-hole insertion task [Lozano-Pérez *et al.* 1984]. The goal of this task is to move a point robot to the bottom of the hole, using compliant motion; there is uncertainty in motion direction and position sensing. We would like to compute a set of initial starting positions from which the illustrated commanded motion direction will succeed in moving the point robot to the bottom of the hole; in this example, the robot reaches the starting position by lowering itself into the plane. The 100% contour in the figure is the result of applying the usual strong planning algorithm. The 60% contour was computed by

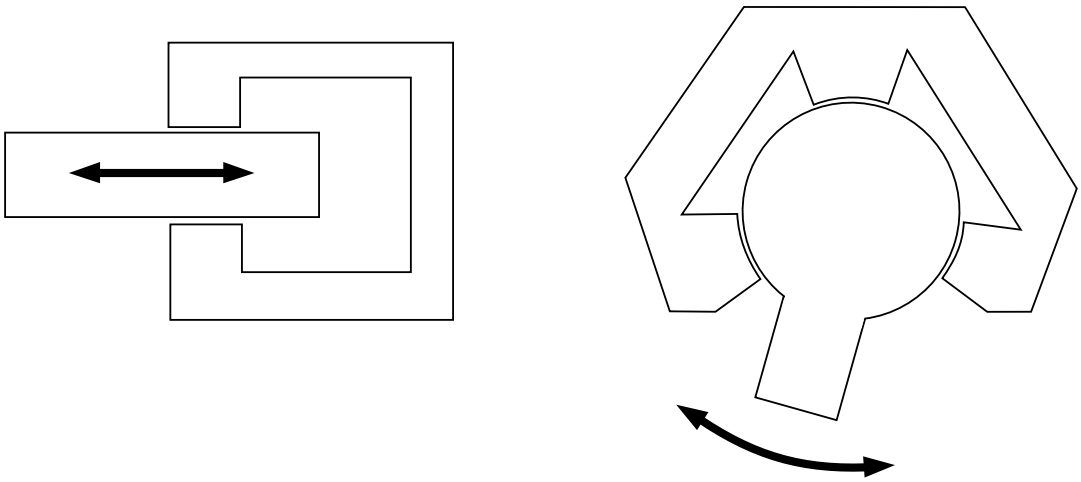


(a) (b)

Figure 28: Situations where computing plans with reduced uncertainty estimates can significantly increase the size of the constructed set. (a) Probabilistic plans constructed using the same method as in Figure 27(a), but assuming a clipped Gaussian error distribution. This distribution allows increased expansion for a given reliability reduction. (b) Coupling between friction uncertainty and motion direction uncertainty. In this example, sticking occurs with probability 0.006; neglecting this possibility removes a constraint that severely limits the set of successful starting positions.

reducing the direction error ϵ_δ by 23% and the position error ϵ_{xy} by 12%, and applying the same strong algorithm. The estimated success probability was then computed by realizing that if the distribution of errors is uniform, the true direction has a 0.77 chance of being within the reduced limits, and the true position has a 0.77 chance of being within the reduced error radius (0.88^2). The contour was computed under the assumption that both parameters are within the reduced limits; this is true with probability $0.77 \cdot 0.77 \approx 0.60$.

The 60% contour in Figure 27(a) is not substantially larger than the 100% contour, even though the estimated reliability has dropped 40%. This is because we selected a particular subrange of the uncertainty parameters that would occur with probability 0.6, and computed the initial positions that would succeed for that subrange. Since there are uncertainty conditions outside the subrange that might also lead to success, 60% is a lower bound on the true success probability. There are other choices of subranges that also occur with probability 0.6, and these give rise to similar sets of starting positions. To construct the maximal set of starting positions that will succeed with probability ≥ 0.6, we must choose each possible subrange, compute the corresponding set of successful starting positions, and form the union of all of the resulting sets. Figure 27(b) shows this union for the probabilities 0.0, 0.5, and 1.0. These hand-constructed contours show the set of all initial starting positions that will succeed with the associated probability; points outside the contour have strictly lower success probabilities.



Prismatic Revolute

Figure 29: Kinematic lower pairs in the plane. Revolute lower pairs cannot be represented without curved edges.

The 60% contour shown in Figure 27(a) is clearly a very conservative estimate of the maximal set shown in Figure 27(b), which suggests that the approach described above will not yield substantially increased action sets when the desired reliability is reduced. This conclusion is premature. In some situations, the approach does produce significant gains for small changes in reliability; Figure 28 shows two examples. Further, good overall performance may still be obtained over a sequence of actions even if the estimated reliability of a single action is significantly less than 100%; see [Erdmann 1989] and [Goldberg 1990] for examples. This discussion has underscored the possible advantages and difficulties of adding probability information to the algorithms described in this thesis. Obviously, much further work is required.

Expanding the class of possible objects. The algorithms in this thesis assume that the objects in the task are polygons with linear edges, and the shape of the polygons is precisely known. Relaxing these assumptions would allow us to expand the scope of the planning algorithms to include a broader range of manipulation tasks.

One possible extension is to allow polygons that have curved edges as well as linear edges. This extension would allow representation of circular objects and fingers, which have some useful inherent stability properties in grasping operations [Cutkosky 1985]. Further, curved edges are necessary to represent revolute joints, which are one of two lower pairs required to represent planar mechanisms (Figure 29). While these advantages provide an incentive to extend the algorithms to include curved edges, substantial work is required to fill in the details. The enumeration of possible configuration-obstacle features must be extended to include facets corresponding to edge-edge contacts that have two degrees of freedom; the only facets currently possible are vertex-edge and edge-vertex contacts. This new facet implies several new types of obstacle edges, which arise wherever facets intersect. For each of these new features, the necessary topological and geometric analysis must be worked out.

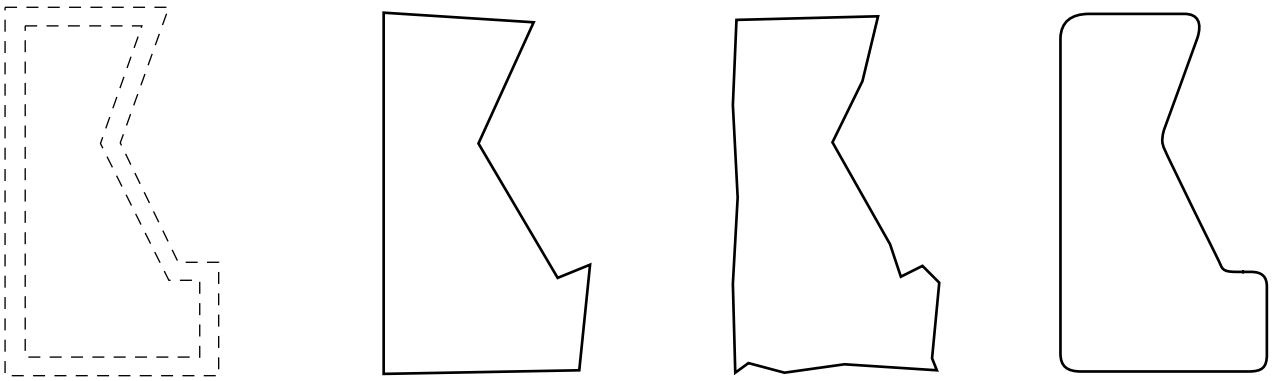
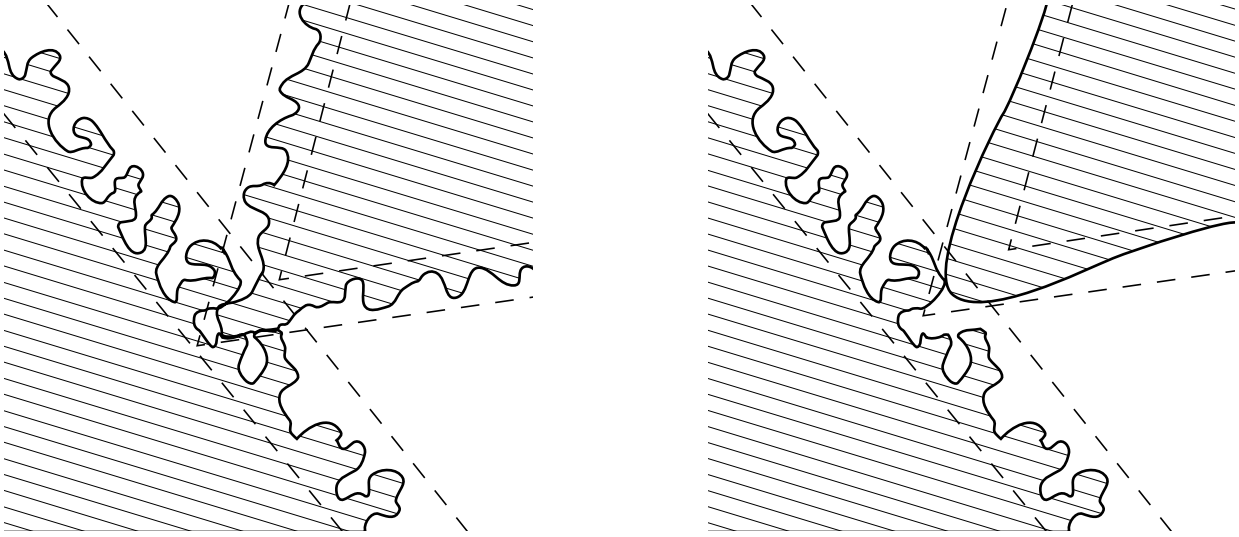


Figure 30: Defining an object with shape uncertainty. The true shape of the polygon may correspond to any simple closed curve contained between the inner and outer shape limits. Some possible true shapes are shown.

A more important extension is to allow uncertainty in the object shapes. Currently the algorithms allow uncertainty in every physical parameter except shape; this exception severely restricts the scope of the planning algorithms. Past work has explored explicit representations of shape uncertainty, where every admissible variation in shape is represented by an explicit parameter [Donald 1987a]. Another approach is to implictly represent shape uncertainty, using boundaries that constrain the extent of the possible shapes. Figure 30 shows a possible choice of constraining boundaries.

This representation of shape uncertainty requires us to extend our representation of the set of possible contact configurations. This set may still be expressed as a set in the (x, y, θ) configuration space, but the familiar configuration-obstacle surface is "blurred" by the presence of shape uncertainty. The facets of the blurred obstacle are volumetric slabs instead of two-dimensional surfaces, and the obstacle edges and vertices are also volumes corresponding to the intersection of two or more slabs. The interpretation of points in this configuration space is non-deterministic: points outside the configuration obstacle are configurations where there is definitely no contact, points strictly inside the configuration obstacle are illegal configurations where the objects definitely inter-penetrate, and points within the obstacle feature volumes are configurations where several contact conditions are possible, including no-contact and all combinations that include or exclude the contact-pairs associated with each slab that intersects the obstacle feature.

The toplogy of this blurred configuration obstacle is complex, and computing an exact representation of this set may be very difficult. However, it may be possible to construct useful approximations to the blurred obstacle. One approach would be to construct a polygon representing the innermost limit of the shape boundaries of each object, use these polygons to construct the known-shape configuration obstacle, and then use an appropriate distance metric to determine all of the applicable slabs for a given configuration-space point. This would provide an implicit representation of the blurred configuration obstacle. A second approach might be to develop a voxel representation, where occupied cells are labelled with free, occupied, or mixed flags; the mixed cells would also have a list of all of the slabs that intersect the cell, so that contact information could be derived.



(a) (b)

Figure 31: The mechanics of interacting uncertain shapes. (a) Since any shape consistent with the shape boundaries is possible, microscopic variations in shape can significantly affect the system mechanics. (b) If the robot's end-effector is known to have no vertices sharper than a certain minimum radius, microscopic variations in the object surface can only have a limited effect on the contact mechanics.

The proposed representation of shape uncertainty also requires an extension of the current mechanics analysis. This is because the object surface properties are underdetermined by this model. Since we have only asserted that the true object shape is bounded by two extremal concentric curves, any local shape is possible; Figure 31(a) shows an example. This weak shape model admits the possibility of pathological microscopic interactions between surfaces which give rise to infinite friction coefficients — two-dimensional "velcro," as it were. In some tasks there may be so much uncertainty that this is an appropriate model, but in other tasks we may have information describing the smoothness of the objects. This information may be available from models of the objects and their surface texture, knowledge of the manufacturing process used to fabricate the objects, history information obtained during past manipulation actions, or knowledge of the minimum radius of curvature of the robot's end-effector (Figure 31(b)). Regardless of the source, this information can be used to bound the set of possible contact normals for a given contact point, which in turn modifies the effective maximum coefficient of friction. Thus microscopic surface irregularities can be included in a macroscopic model of the object's shape.

Modeling more complex task spaces. With the exception of the two-finger grasping example described above, all of the tasks addressed thus far can be analyzed in a three-dimensional (x, y, θ) space. This space is not sufficient for all tasks; some tasks must be analyzed in a higher-dimensional space, due to the intrinsic nature of their physical systems. Additional dimensions may be required for a variety of reasons:

- The task may require more configuration variables. In general, three variables are required to specify the position of a planar body, six variables are required to specify the position of two planar bodies, nine variables for three bodies, and so on. In this thesis, we are able to represent the configurations of a two-body system using only three variables; this is possible because the relative configuration of the bodies is sufficient for analysis of the task mechanics.

  This is not the case for some tasks. For example, to synthesize pushing actions where the support surface has regions of varying friction properties, the object motion will depend on the position of the object on the support surface, as well as the relative configuration of the object and the pushing finger. Thus a pushing task with a non-uniform support surface requires a six-dimensional configuration space, unless we use a very conservative mechanics model that assumes that the object is anywhere on the support surface. More than three configuration variables will also be required if additional objects are present. For example, pushing or grasping in the presence of obstacles requires six configuration variables, because the world position of the manipulated object determines whether it will collide with an obstacle during the operation, which in turn affects the object's motion.

- The task may require an extended notion of state. The (x, y, θ) space provides a kinematic description of the state of the task; more information may be required to adequately predict the behavior of the task's physical system. For example, dynamic properties may be significant, requiring velocity terms to be included. In these situations the configuration space must be extended to a phase space that includes a velocity term for each configuration variable, which doubles the number of required dimensions.

  It may also be necessary to join the configuration space with other parameter spaces at certain configurations. For example, we shall see in Part II that the behavior of rigid bodies in contact is ambiguous in certain statically-redundant configurations; we may wish to associate internal force parameters with these configurations in order to resolve the ambiguity. A second example is the analysis of tasks involving collisions; these tasks may be analyzed by treating the configuration-obstacle surface as a Poincaré surface that joins the configuration space with an impulse space describing the change of momentum developed during a collision event [Wang 1989].

- The task is not planar. This thesis assumes that the task objects are planar, and that all forces and motions are confined to a plane. Real manipulation tasks are three-dimensional, but under certain conditions this planar model provides a reasonable and practical approximation of the task. These situations include tasks involving prismatic objects resting on a flat or inclined support surface, where the height of the objects is small compared to their width. However, other tasks may not satisfy these conditions.

  We can extend the scope of this work by addressing nonplanar tasks. A simple extension would be to add a third translational dimension, but only allow rotations about a single axis. This extension would give us an (x, y, z, θ) configuration space, which corresponds to the workspace of many SCARA-type manipulators found in industrial assembly lines. We would then need to extend our algorithms to construct plans in this space. One approach

would be to restrict the class of admissible objects to collections of stacked polygonal prisms, with the axis of each prism aligned with the z-axis. This restriction would allow us to represent the task configuration space with a discrete collection of (x, y, θ) configuration spaces, each with an associated interval of z-values. Additional work would be necessary to avoid tipping moments that cause unacceptable rotations. A second appproach would be to again assume that rotations are only possible about the z-axis, but include objects that are not prismatic, such as pyramids or tetrahedra. Accurately modeling the (x, y, z, θ) configuration space for these objects will require a continuous representation that captures the coupling between lateral and vertical displacements. An even more difficult extension would be to allow general three-dimensional motion. This requires a six-dimensional configuration space, with six-dimensional configuration obstacles. See [Donald 1987b] for methods of representing and constructing these obstacles.

These issues are examples of features that add complexity to a task, requiring a more complex space for task analysis. Possible directions are outlined for addressing some of these issues, but much future work remains.

Unlikely Extensions

The above discussion focused on limitations of the work presented in this thesis, and methods for removing some of these limitations. Some of the proposed extensions are difficult, but with enough hard work and clever insights, we may eventually find practical solutions to these problems. In contrast, this section considers problems that appear to be fundamentally outside the scope of the methods presented in this thesis; these problems probably require a completely different approach.

Manipulating aggregates of objects. In some tasks, we may wish to manipulate large numbers of objects simultaneously. For example, we may wish to sweep a pile of coins off the edge of a table into our hand, or grab a handful of popcorn from a bowl. In these tasks, there are so many moving objects that it seems hopeless to model the corresponding high-dimensional task configuration space. More to the point, we probably don't *care* about the detailed motion of the individual objects, as long as suitable aggregate behavior is achieved.

Deformable objects. Our configuration-space algorithms rely on the assumption that the shape of the objects is fixed. If we extend our representation to include shape uncertainty, it may be possible to also model small shape deformations within this framework. However, gross shape deformations seriously undermine one of the key properties exploited by the algorithms: that the configuration-space obstacle is constant. This property does not apply to tasks involving modeling clay, soil, liquids, rope, fabric, and other flexible or amorphous materials. Effective manipulation of such objects may require consideration of other task properties, such as conservation of volume, the behavior of flexible bodies suspended in a gravity field, elastic and plastic deformation properties of materials, the angle of repose of a powdered material, and so on.

Control laws with state history. The mechanics algorithms presented in this thesis rely on the assumption that the physical behavior of the task can be predicted using only information that describes the task state. In the current algorithms, this state is represented by the task configuration, but future extensions may also include velocities, internal forces, and other parameters to

describe the state of the task. With or without these extensions, the analysis algorithms assume that the combined behavior of the robot control law and natural task mechanics may be predicted using only the current state of the physical system.

What if the task behavior also depends on the history of previous states as well as the current state? Computer-controlled robots can easily produce such behavior, since the controlling program may use past sensory information to guide control decisions. Such control laws may increase the effectiveness of the manipulation system by performing an on-line constraint analysis during task execution. Our planning algorithms would not be able to construct plans for these controllers, however, since the task state is no longer sufficient to predict the system behavior. In some situations it may be possible to extend the task state-space to include the state of the controller, but this is infeasible for general control laws utilizing sensor history information. There has been substantial past work exploring this topic; see [Lozano-Pérez *et al.* 1984; Mason 1984; Erdmann 1986; Natarajan 1988] and [Canny 1989] for examples.

* * *

So what have we learned? From the experimental results obtained thus far, the proposed approach to planning reliable manipulation actions appears to be feasible for planar tasks involving two polygonal objects with configuration-dependent task mechanics. For these tasks, most of the required analysis and implementation work has been completed, resulting in programs and physical experiments that support the theoretical analysis. However, some questions remain unresolved, and should be addressed in future work.

Looking ahead, we have also examined ways to extend the scope of these results. Extensions to include two-fingered grasping tasks and controllers with a sense of time appear to be within reach. For other extensions the prognosis is less clear, but we have some leads on how to proceed; these include adding multi-step or probabilistic actions, allowing objects with curved edges or uncertain shapes, and modeling higher-dimensional spaces to accomodate the presence of other objects, more complex task dynamics, or non-planar motion. Finally, we have seen tasks that appear to be fundamentally outside the scope of this approach; these tasks include manipulating large numbers of objects, manipulating deformable objects, or planning actions for control laws that employ a general sense of history.

So the approach has been successful so far, and can likely be extended to include tasks of at least slightly higher complexity. The approach also has clear limitations; these suggest that, like many other analytical tools, the methods and algorithms developed in this thesis will be most useful for a particular domain of problems, and unsuitable for others. Future research will further illuminate these issues.

In this section, we have assessed the current status of this work, and considered the implications of this thesis for future research. It is equally important to look into the past, and see how this work is related to previous results which inspired it; developing this context will further clarify the contribution of the thesis to the field of robotics research. This is the topic of the next section.

Relationship to Previous Work

This thesis draws on the results of many previous authors, and is especially inspired by three major lines of research:

- Geometric approaches to robot planning problems.
- Analysis of manipulation task mechanics.
- Modeling and compensation for task uncertainties.

The thesis attempts to bring together results from each of these areas to produce a unified approach to the analysis and planning of planar manipulation tasks. We will now review the past work in these areas, and emphasize the previous developments that led to this thesis. We begin with some of the early robotics work that helped to identify these important problem areas.

Seminal Work

The earliest robotics efforts noticed the significance of task mechanics in the execution of manipulation actions [Ernst 1961; Goertz 1963]. These authors remarked on the difficulty of certain tasks due to limited control precision, and suggested alternative control schemes to ameliorate these problems. One of the principle suggestions was to extend the manipulator's controller to explictly measure and respond to force information.

Inoue [1974] used a force-controlled robot arm to automatically assemble a collection of tight-fitting parts. Inoue exploited the arm's force control to develop strategies that succeeded in assembling parts with tolerances much finer than the arm's positioning resolution. As a part of these strategies, Inoue explicitly considered the effects of uncertainty on the executed actions, and modified his strategies to succeed in the presence of uncertainty. In so doing, Inoue developed strategies such as squeezing a washer multiple times to center it between the robot's fingers, or tilting a peg to increase the amount of acceptable position uncertainty. These techniques combined with the force control capabilities of the manipulator to produce successful actions. All of Inoue's strategies were hand-designed and programmed. Similar strategies were also evident in other contemporary experiments that demonstrated the assembly prowess that can be attained by carefully-designed manipulation systems [Pingle *et al.* 1974; Bolles and Paul 1973].

Meanwhile, other researchers attacked the problem of automatic planning. Perhaps most relevant among these were the BUILD and STRIPS programs [Fahlman 1974; Fikes and Nilsson 1971]. These programs performed analysis in an abstracted blocks-world, where actions were represented by logical state changes. In the BUILD program, actions corresponded to instaneous position changes; a manipulated block was assumed to disappear from one position and reappear in another position. Given a desired final configuration of blocks, the BUILD program identified a sequence of these actions that would produce the desired final configuration, while also assuring that each intermediate state corresponded to a stable configuration of the blocks. The STRIPS program represented actions as precondition/postcondition pairs, and assumed that if the action preconditions were met, then execution of the action would produce the action postconditions. These conditions were expressed in abstract geometric terms, such as `INROOM(BLOCK-A,ROOM-1)`. While the plans generated by the STRIPS program were executed on a real robot, the low-level issues of control and geometry were left as engineering exercises for the designers of the experimental apparatus. A third example in this genre was the HACKER program, which also considered simple blocks-world problems [Sussman 1973].

While these programs deliberately ignored the details of task mechanics and uncertainty, they did illuminate a variety of important and difficult issues involved in planning solutions to complex problems. These problems included the interaction of plan subgoals, control of search procedures, and the selection of suitable problem representations. As a result, these investigations defined a research agenda for addressing problems that can be expressed via symbolic representations [Newell and Simon 1976]. Much work has followed these seminal results, and we will not review it here. However, the MOLGEN planner is especially relevant because the constraint-satisfaction methods it introduced are analogous to our analysis of physical constraints inherent in manipulation tasks [Stefik 1981].

Not all work in automatic planning was purely symbolic. Lozano-Pérez [1976] addressed the manipulation planning problem, and attempted to bridge the gap between the continuous domain of task geometry and the symbolic domain of other planners. Lozano-Pérez identified several geometric sets that can be automatically constructed from a description of the task geometry. Successful actions may then be identified by constructing various transformations and intersections among these sets; the semantics of these sets are analogous to the preconditions and postconditions of the symbolic planners. In a similar vein, Taylor [1976] addressed the problem of automatically converting high-level task descriptions into detailed robot programs. Using representations of the task goals, object geometry, object locations, and other descriptive parameters, Taylor developed a frame-based program that selected relevant manipulation strategies from a database of canonical strategies, and then constructed the necessary missing information required to convert the canonical strategy into a concrete plan. Taylor's work focused especially on the problems of representing and manipulating task uncertainties.

These early results set forth an agenda for manipulation planning research. It became clear that task mechanics and uncertainty play a critical role in real-world manipulation tasks, and analysis of task geometry is essential to solving general manipulation problems. Thus the problem of automatic manipulation planning became one of identifying ways of coping with task geometry, mechanics, and uncertainty. Further, the results obtained in domain-independent planning research showed that symbolic task representations may be used to address high-level planning issues; if the low-level methods developed to plan manipulation strategies could eventually be converted to symbolic representations, then these results may also be brought to bear.

Efforts that followed addressed various chunks of this agenda, and this thesis is no exception. The discussion below will group these efforts into investigations of task geometry, mechanics, or uncertainty; however, this grouping is primarily for convenience, since many of the examples simultaneously addressed combinations of these areas.

Geometric Approaches

Several researchers have focused on the geometric aspects of manipulation tasks. The problem that has received the greatest attention is the *path-planning problem:* Given a robot placed in a world containing obstacles, how can we synthesize a path to a desired position that avoids collisions with the obstacles? There have been several approaches to this problem. One of the early breakthroughs in the area was the application of configuration space [Udupa 1977; Lozano-Pérez and Wesley 1979]. The notion of configuration space was developed in the field of kinematics for the analysis of mechanisms [Reuleaux 1876], and has a direct relationship to the problem of analyzing robot motions in the presence of other objects. The configuration-space approach to robot path planning has received considerable attention over the years, leading to several planners that operate by manipulating configuration-space sets [Brooks and Lozano-Pérez 1985; Donald 1987b; Lozano-Pérez 1987]. Further, the characterization of these sets has become a research area in itself, and several researchers have developed exact representations of configuration-space obstacles for various domains [Bajaj and Kim 1988; Avnaim *et al.* 1988a; Avnaim and Boissonnat 1989; Brost 1989; Ge and McCarthy 1989; Dooley and McCarthy 1990].

Related approaches to the path-planning problem include the spatial decomposition approach, where collision-free positions are grouped into regions that allow certain canonical motions anywhere within the region; these regions are then connected by motions that allow transitions from one region to another. These regions may be identified either in configuration space or the original space [Brooks 1983; Yap 1987; Tournassoud and Jehl 1988]. A different approach is the potential-field approach, where an energy function is defined that attracts the robot to the desired position, but repels it from obstacles. This heuristic technique may give rise to very efficient computation of paths, but is sometimes thwarted by local minima in the energy function. Current research is attempting to remove this limitation. As with the spatial decomposition approach, potential energy path-planners have been formulated both in configuration space and the original space [Hogan 1985; Khatib 1986; Rimon and Koditschek 1989; Barraquand *et al.* 1990; Volpe and Khosla 1990]. Finally, some researchers have addressed the inherent complexity of the general path-planning problem, showing that the complexity of complete solutions must grow exponentially in the number of degrees of freedom, and polynomially in the environment complexity [Schwartz and Sharir 1983; Canny 1987].

Other geometric problems have also been addressed. For example, researchers interested in developing qualitative techniques for analyzing physical systems have studied the kinematics of mechanisms and other physical systems using configuration-space constructions [Faltings 1987; Joskowicz and Addanki 1988; Joskowicz and Sacks 1990]. Other researchers have developed grasp-planning algorithms that use geometric constraints to identify a set of grasp configurations that are feasible at the initial and final stages of a manipulation task; these constraints include the presence of obstacles and robot workspace limits [Lozano-Pérez 1981; Lozano-Pérez *et al.* 1988; Jones and Lozano-Pérez 1990].

The above researchers developed methods for solving manipulation problems by constructing and manipulating geometric sets. Other researchers concurrently attacked the problem of developing computer representations for these sets, and algorithms to perform the necessary transformations. The field of computational geometry is very large, and we will not present a thorough review of it here. However, two especially relevant components of the field are the development of computer representations of three-dimensional polyhedra, and the more recent study of methods to improve the robustness of geometric computations. As we shall see in Part II, both of these developments strongly affect the *CO* algorithm used to construct configuration-space obstacles. Seminal work on representing polyhedra includes [Baumgart 1974; Requicha 1980; Wesley *et al.* 1980; Wesley 1980; Fitzgerald *et al.* 1981]; see [Milenkovic 1988], [Hoffmann 1989], and [Sugihara and Iri 1989] for excellent reviews of the issues related to robust geometric calculations.

Exploiting the Task Mechanics

The mechanics of a manipulation task decide whether the performed action will succeed. This basic observation has motivated a terrific amount of manipulation research, which we will only briefly review here. For a discussion of the general relationship between task mechanics and manipulation actions, see [Mason 1985]. We will proceed by examining several of the areas that have been investigated.

Compliant Motion. The earliest research in manipulation task mechanics focused on force control and compliant motion. This illuminated two important problems: The design of force-controlled manipulators,and the automatic transformation of task goals into force-controlled motion commands. Early design results include [Whitney 1977]; see [Whitney 1987] for a survey of force control schemes. In order to relate task requirements to force-controlled motion commands, Mason proposed a method that used task constraints to divide the control freedoms into position-controlled and force-controlled subspaces [Mason 1981]; an implementation of this hybrid force/position control scheme was used to demonstrate peg insertions and other tasks [Raibert and Craig 1981]. Several other methods have been proposed for converting task requirements into force-controlled motion commands; we will review some of these in the next section.

Peg Insertion. Because of its significance in assembly applications, the task of inserting a peg into a hole in the presence of control errors has received much attention. Several force-control schemes were developed specifically to address this task; see [Goto *et al.* 1980] for an example. Other researchers performed an analysis of the task mechanics in order to identify robust manipulation strategies; these efforts have vastly improved our understanding of this important manipulation task, and ultimately led to the development of simple passive mechanisms that allow open-loop insertion of pegs with very tight tolerances and complex shapes [Simunovic 1975; Drake 1977; Whitney 1982; Sturges 1988; Strip 1989; Caine *et al.* 1989]. In addition to solving an important class of manipulation tasks, this work established a paradigm for solving manipulation problems via careful analysis of the task mechanics.

Friction and Force Modeling. The work on peg insertion also underscored the importance of contact friction in manipulation tasks. Friction has been studied for hundreds of years; significant advances in our understanding of friction began in the eighteenth century [Coulomb 1781]. Since that time, a series of abstractions have been developed to simplify the analysis of problems involving

multiple frictional contacts [Moseley 1835; Reuleaux 1876; Ohwovoriole *et al.* 1980; Ohwovoriole 1980; Erdmann 1984; Rajan *et al.* 1987; Brost and Mason 1990]. Further, analysis of frictional contact problems is complicated by the presence of ambiguities and inconsistencies inherent to the Newtonian model of rigid bodies interacting in the presence of Coulomb friction; see [Featherstone 1986] and [Mason and Wang 1988] for discussions of these problems. These abstractions and ambiguities play a central role in the analysis of manipulation task mechanics developed in this thesis; we will review these results in detail in Part II.

Pushing. As shown above, pushing actions provide a robust method for orienting, acquiring, or repositioning objects resting on a flat surface. Predicting the motion of a pushed object is difficult, because of the ambiguities inherent to rigid-body mechanics with Coulomb friction, and due to uncertainty in the distribution of support forces between the object and the supporting surface. The motion of a body sliding along a support surface has been studied for many years, and we draw upon these results in the pushing mechanics analysis presented in Part II [Jellet 1872; Prescott 1923; MacMillan 1936; Mason 1986a; Mason 1986b; Peshkin and Sanderson 1988a; Goyal 1989]. Further, other researchers have applied these basic mechanics results to analyze manipulation tasks involving pushing. These efforts have produced planners that synthesize pushing motions for an infinite fence in the presence of position and control error [Mani and Wilson 1985; Brost 1988], a planner that constructs a sequence of fence-pushing operations to unambiguously orient a part [Peshkin and Sanderson 1988b], a technique for analyzing two-finger pushing operations that remove uncertainty in a part's position and orientation [Balorda 1990], and a method for analyzing high-speed pushing operations that include dynamic effects [Pham *et al.* 1990].

Grasping. The problem of selecting and achieving a useful grasp of an object is obviously an important manipulation task. Therefore it is not surprising that many researchers have studied various aspects of this problem [Salisbury and Craig 1982; Chelpanov and Kolpashnikov 1983; Abel *et al.* 1985; Baker *et al.* 1985; Cutkosky 1985; Wolter *et al.* 1985; Barber *et al.* 1986; Kerr and Roth 1986; Jameson and Leifer 1987; Mishra *et al.* 1987; Li and Sastry 1988; Montana 1988; Markenscoff and Papadimitriou 1989; Park and Starr 1990]. Each of these papers addresses some aspect of the mechanics of grasping; topics include grasp kinematics, frictional stability of grasps, synthesis methods for certain classes of grasps, and the theoretical existence of stable grasps. For a survey of these and other grasping results, see [Pertin-Troccaz 1989]. While most of the past work in grasping is related to this thesis because of its similar mechanics analysis, a few papers are especially relevant. Nguyen [1988] develops the notion of *force closure,* which plays a significant role in the force analysis presented in Part II. Salisbury [1982] presents a taxonomy of three-dimensional contacts that is similar to the planar contact taxonomy analyzed in this thesis. Finally, several authors developed analysis and planning methods which consider finite motion of the object or fingers during the grasp operation; these finite motions play a crucial role when actions are executed in the presence of uncertainty [Hanafusa and Asada 1977; Fearing 1986; Mason and Brost 1986; Brost 1988; Trinkle *et al.* 1988; Goldberg 1990].

Analysis of Fixtures. As mentioned previously, industrial automation systems typically make use of special-purpose fixtures for orienting and constraining parts. Such fixtures are widely used, and methods for automatic analysis of fixtures may yield substantial cost savings by streamlining the development of automation systems. For parts-orienting fixtures, Boothroyd *et al.* [1972] addressed the statistical distribution of rest configurations of parts, in an effort to aid the design of automatic parts feeders. Grossman and Blasgen [1975] developed a general-purpose parts feeder that uses

a combination of passive mechanics and active sensing to disambiguate the rest state of a fed part. The computational aspects of part feeder design have also been addressed [Natarajan 1989]. For fixtures intended to constrain a part's position, several authors have developed programs that automatically design fixtures [Asada and By 1985; Englert and Wright 1986; Strip and Maciejewski 1990]; this problem is closely related to the grasping problem.

Assembling Snap-Fit Parts. In an unique extension of conventional configuration-space methods, Donald and Pai [1990] developed an algorithm for predicting the motion of an ensemble of rigid planar parts connected with spring-loaded hinges; these ensembles approximate the "snap-fit" parts commonly found in commercial products. Using an extension of conventional configuration-space representations and an analysis of friction, their program predicts the motion of the parts during a specified assembly operation. This program may be used as an aid during the design of snap-fit assemblies, as well as a tool for predicting the behavior of parts during assembly processes.

Collisions. Many of the mechanics analysis results described above assume that all motions are quasi-static; that is, frictional forces dominate inertial forces. As long as motion speeds are slow and there is significant ambient damping, this is a reasonable assumption. However, some manipulation tasks are highly dynamic, and involve high-speed collisions between the manipuator and the manipulated object. This reality motivated several researchers to study the analysis and control of collisions in manipulation tasks [Brach 1984; Wang and Mason 1987; Wang 1989; Buhler *et al.* 1990]. This thesis also addresses manipulation actions involving dynamic collisions (the placing-by-dropping task), but does not utilize these results because they require velocity information. Instead, the thesis applies a weak mechanics model and uses the task kinematics to identify a set of successful initial dropping positions. This approach is related to the qualitative inferences produced by the FROB system for a simpler geometric domain [Forbus 1981].

Simulation. The BP_i algorithm may be viewed as a numerical simulation of the manipulation task mechanics. There is a substantial amount of previous work devoted to the development of computer simulations of mechanical systems; we will not review these results here. However, a few previous simulation systems are especially relevant to this thesis, because they address the interaction of two or more rigid bodies under various dynamical conditions [Lötstedt 1981; Featherstone 1986; Hoffmann and Hopcroft 1986]. As mentioned above, some of the anomalous situations that challenged these systems similarly affect the BP_i algorithm.

Grappling with Uncertainty

The earliest researchers in robotics noticed the significance of uncertainty in manipulation tasks, and substantial effort has been devoted to developing systems that are robust in the face of uncertainty. There have been several approaches to this problem, but one is especially relevant to this thesis. We will review this most relevant approach first, and then discuss other approaches.

The LMT Approach

The approach that is most closely related to this thesis is the formalism developed by Lozano-Pérez *et al.* [1984] for planning compliant motions in the presence of uncertainty. This approach considers uncertainty in sensing and control, and models each source of uncertainty with a worst-

case error bound describing the maximum possible discrepancy between the robot's estimate of a parameter and the true parameter. Under the assumption of generalized damper dynamics, the approach identifies sets of initial positions that will achieve the goal under the execution of a given action (Figure 27 shows an example). These sets are called *preimages.* Multi-step plans may be constructed by recursively constructing preimages backwards from the goal; if any constructed preimage fully contains the set of current possible robot positions, then the corresponding sequence of actions may be executed to reliably achieve the goal.

This formalism provided a foundation for several subsequent studies of the nature of manipulation planning in the presence of uncertainty. The ensemble of all of this work is often referred to collectively as the *LMT approach.* Mason [1984] proved the correctness and completeness of the formalism, and explored the relative power of various motion termination predicates. Erdmann [1986] continued this study of the computational aspects of preimages, and showed that preimages with general termination predicates are not computable for arbitrary environments. Erdmann's proof required a recursive definition of the task environment; Canny [1989] later showed that preimages are computable for more practical environments under the additional assumption that the maximum number of plan steps is specified in advance. The inherent complexity of motion planning in the presence of uncertainty has also been studied using the LMT formalism. Natarajan [1988] showed that the problem is *PSPACE*-hard under a termination predicate with a sense of time, and Canny and Reif [1987] showed that the problem is non-deterministic exponential-time hard under the general termination predicate proposed in the original LMT paper.

These theoretical results were accompanied by the development of efficient algorithms for computing preimages under restricted termination predicates. In the same paper that presents his computability result, Erdmann developed and implemented an algorithm for constructing preimages under a stateless termination predicate; these simplified preimages are called *backprojections.* Briggs [1989] extended Erdmann's algorithm by showing how to efficiently construct backprojections while considering arbitrary motion directions. Latombe [1989] reformulated some of the definitions in the original Lozano-Pérez, Mason, and Taylor paper, extended the previous work by addressing maximal preimages and conditional actions, and presented extensions to Erdmann's backprojection algorithm.

Other authors extended the basic LMT formalism in various ways. Donald [1990] added two significant extensions. In the first extension, Donald showed how variations in the task geometry could be included by adding additional dimensions to the configuration space; preimages can then be constructed in this higher-dimensional generalized configuration space. Donald's second extension was to redefine task success by proposing the notion of error-detection and recovery strategies. In these strategies, the goal of the task is extended to include either recognizable achievement of the goal, or recognizable failure. Recognizable failures allow subsequent recovery actions to be taken; these recovery actions depend on the details of the task and the nature of the failure. These error-detection and recovery strategies allow plans to be constructed even though a guaranteed plan is not available. Erdmann [1989] also extended the formalism by addressing tasks with no guaranteed plans. Erdmann showed how randomized actions may be used to achieve the goal in a finite time, even though sensor uncertainties prevent the construction of guaranteed actions. Buckley [1989] extended the formalism in a different way by addressing tasks with three degrees of translational freedom under generalized spring control, and showing how to construct backprojections for this mechanical system. Brost [1988] addressed a parallel-jaw grasping task, and showed how to construct regions of initial conditions that achieve a desired final configuration. These regions are analogous to LMT backprojections.

This thesis views the preimages and backprojections defined in the LMT approach as examples of goal-achieving sets that exist for any manipulation task. The continuous nature of the physical world assures that these sets exist for tasks with arbitrary task mechanics, provided that the task description is formulated in the appropriate space. I have attempted to develop this notion by constructing backprojections for tasks that have very different physical properties from the generalized damper dynamics previously studied. The sets V_{state} and V_{command} may be viewed as backprojections of the goal under the specified action, but I have refrained from using the terms "preimage" and "backprojection" in the above description because the relationship between these sets and the LMT sets is muddled. For example, which of the two volumes is the preimage? Both sets are constructed in different spaces, and neither are constructed in a space that corresponds exactly to the configuration space defined in the LMT formalism. This distinction may seem academic, but becomes increasingly problematic when multi-step actions are considered that require preimage construction in several different spaces. Despite this ambiguity, I have named the BP_e and BP_i algorithms after the backprojection algorithm developed by Erdmann, because these algorithms are so closely related. Further, it is my hope that the many insights developed while studying the LMT approach will be easily adapted to the methods developed in this thesis. For example, extending the thesis analysis to include error-detection and recovery strategies or randomized actions would allow planning of actions when guaranteed plans are unavailable.

Other Approaches

While the LMT approach to handling uncertainty is the most closely related to this thesis, other approaches have also been successfully investigated. Some of these methods are similar to the techniques applied in the thesis, while others are very different. We will examine each of these approaches in turn.

Identifying Task Invariants. Sometimes it is possible to identify certain structural characteristics of a manipulation task that enable the development of strategies that are robust in presence of tremendous uncertainty. An example of such a strategy is the path-generation algorithm developed by Lumelsky and Stepanov [1987]. This algorithm assumes that the environment geometry is completely unknown, but that a moving robot can sense an obstacle when it is encountered, and also keep track of positions it has visited. Under these assumptions Lumelsky and Stepanov derived a simple motion-generation procedure that guarantees that the robot will eventually reach the goal if it is reachable by any path.

Constraint Propagation. This thesis and the LMT formalism represent uncertainty conditions using explicit geometric sets; it is also possible to represent uncertainty using systems of constraint inequalities. Combinations of these inequalities form an implicit representation of the set of actions that will succeed in the presence of uncertainty. Once these inequalities have been formulated, symbolic manipulation may be used to identify successful actions, or to decide that additional uncertainty-reduction steps are necessary [Brooks 1982]. In a related paper, Taylor and Rajan [1988] also represent task uncertainties using collections of inequalities, but analyze these constraints numerically instead of symbolically. The constraint inequalities implicitly define a polytope in a high-dimensional parameter space, but the effect of these parameters ultimately arises in a task space that has much lower dimension. Special numerical algorithms allow the authors to project the high-dimensional polytope into the low-dimensional task space, which results in a region of acceptable task parameters. This region is semantically similar to the V_{command} volume constructed

in this thesis, since this volume also represents the cumulative effect of many uncertainty parameters expressed in a low-dimensional space. However, V_{command} may be difficult to construct using constraint inequality techniques, since some of the necessary task mechanics cannot be described with closed-form equations.

Constraint Analysis in the Configuration Space. This thesis explicitly represents task constraints as a set of inadmissible configurations $\mathcal{C}$ that should be avoided at all times. Further, the mechanics of a manipulation task imposes contraints on the possible motions of the manipulated object, with associated motion uncertainty. Analogs to these constraints were addressed by a path-planning system developed for mobile robot navigation [Stentz 1990]. This system considers a variety of task constraints, and expresses them in the (x, y, θ) configuration-space of a mobile robot. This space is then searched for a path that reaches a desired destination while satisfying all constraints. The representations and algorithms employed in this planner are very different from those presented in this thesis, primarily due to differences in the underlying structure of the manipulation and navigation problems. For example, configurations in free space are of greatest interest in the navigation problem, while configurations on the configuration obstacle surface should be avoided. In contrast, manipulation tasks often specify obstacle surface points as goal configurations, and motion along the surface is required to remove task uncertainty. An interesting avenue for future work would be to identify the common ground between these two results, in an attempt to construct a single planner that treats both tasks equally well.

Search. In some manipulation tasks, uncertainty may be overcome by executing a sequence of actions that progressively reduce uncertainty. The LMT formalism provides a vehicle for constructing such multi-step actions, but other approaches have also been explored. For example, given an analysis of task mechanics and sensing, planning multi-step tray-tilting actions may be reduced to the problem of searching an AND/OR graph [Erdmann and Mason 1988; Taylor *et al.* 1988]. Peshkin and Sanderson [1988b] also developed a method for constructing sequences of pushing operations using search procedures. Each of these examples demonstrated planners capable of constructing a sequence of actions that unambiguously orient a part starting from a completely unknown orientation. Goldberg [1990] developed a related algorithm that constructs sequences of parallel-jaw grasping actions to orient parts. By exploiting monotonicity properties that are present in certain manipulation tasks, Goldberg's algorithm directly constructs an action sequence to orient a planar part, regardless of its shape.

Statistical Representations. Most of the work reviewed so far has assumed that uncertainty conditions may be modeled by bounded errors. Other researchers have applied statistical models of uncertainty; these models have the advantage that they explicitly model the continuous reduction in uncertainty that may be obtained over a sequence of uncertain actions or sensor measurements. Several approaches have been successfully developed, including entropy models of the manipulation task state [Sanderson 1984], position representations using multivariate Gaussian distributions [Smith and Cheeseman 1986], direct analysis of the probabilistic effect of grasping actions [Goldberg and Mason 1990; Goldberg 1990], grid-based representations of probabilistic safe-navigation regions [Elfes 1987], and Kalman filtering techniques for refining uncertain sensor interpretations [Matthies and Shafer 1987; Ayache and Faugeras 1988]. As mentioned in the previous section, including statistical information could enrich the planning algorithms presented in this thesis, but the required analysis remains a difficult problem.

Learning. Observations may also be used to identify actions that are robust in the presence of uncertainty. Several researchers have investigated methods of using observations to improve task performance in the presence of sensing and control errors; examples are found in the fields of adaptive control, robot control, and machine learning. For a representative sample of these results, see [Dufay and Latombe 1984; Dunn and Segen 1988; Aboaf *et al.* 1989; Christiansen *et al.* 1990; Gross forthcoming]. This thesis does not employ learning techniques, but they may be included in the D function used by the BP_i algorithm. This function describes the task's differential behavior, and may be derived using either analytical or empirical techniques. Empirical learning techniques are especially attractive when analytical approaches are either intractable or overly conservative.

Sensor Fusion. Another approach to reducing the effects of uncertainty is to combine information from several sensors, each of which has its own particular uncertainty characteristics. Interpreting the sensor measurements in combination may result in total uncertainty that is much less than the uncertainty in any individual sensor. Examples of this approach include [Ikeuchi *et al.* 1986; Stansfield 1988; Allen 1988; Durrant-Whyte 1988].

Restricting the Set of Admissible Tasks. Desai [1989] developed an execute-verify strategy that executes compliant motions to solve a restricted class of manipulation tasks. Desai's strategy identifies the topological contact situation attained at the end of a compliant motion, and then uses this information to construct an error-correcting motion. Desai points out that this strategy will fail in certain geometric situations, and provides heuristics for preventing these failures by redesigning the task geometry. Desai proposes the notion of *contact formations,* which are topological analogs to the configuration-obstacle features constructed in this thesis.

Qualitative Reasoning. In some tasks, the relevant properties of the task physical system may be inferred without performing a detailed metric analysis. In the field of qualitative physics, programs have been developed that analyze and control physical systems using models that include only qualitative information, or qualitative information combined with limited metric information. These systems implicitly include uncertainty by identifying system characteristics that are insensitive to fine changes in metric properties. See [Forbus 1988] for a survey of qualitative physics research.

Other Related Work

There is other previous work that is related to this thesis, but which does not fit into the above three categories. We will mention these briefly here.

Because this thesis employs many geometric constructions, it draws substantially from previous work in geometric analysis. As we shall see in Part II, the configuration-space obstacle of two interacting polygons is an ensemble of ruled surfaces. We expoit the intrinsic geometry of ruled surfaces when developing the *CO* algorithm. We also utilize concepts from projective geometry when representing sets of planar forces on the force sphere. See [Hilbert and Cohn-Vossen 1932] for an excellent discussion of the geometry of ruled surfaces, spherical projections, and other geometric constructions.

The dynamic analysis presented in this thesis is related to the control of mechanical systems with time-varying topology. These topological variations may be present in the system control law, or in the mechanical system itself. For example, in variable-structure control, regions of a state space are

identified that are subject to different control laws. None of the individual control laws are stable, but the ensemble of control laws is stable. The control laws are chosen so that ideal control will cause the system to converge on a boundary between regions, and then slide along the boundary to a stable final state. Since real control involves finite switching times, physical implementations of these controllers tend to chatter back and forth across the boundary while converging to the stable state. In this thesis, we study physical systems that have analogous properties. The behavior of a pair of interacting polygons is often unstable for states on a configuration-obstacle facet, but stable for states at the intersections of facets. In some cases, the system state even slides along a boundary between facets before finally stabilizing, as in Figure 19. Since the switching between task dynamics is performed by nature, the chattering evident in digital varible-structure control generally does not occur. For a discussion of variable-structure control, see [Young 1978]. A related topic is the design of control laws for physical systems that have discrete modes of physical behavior, such as mechanical linkages driven by gears with backlash. See [Brockett 1984] for remarks on the control of such systems.

Lastly, the BP_i numerical integration algorithm bears a resemblance to Dijkstra's classic algorithm for finding the shortest path through a discrete graph. In Dijkstra's algorithm, a moving horizon is defined which progressively expands until the destination node is included; this expansion is organized so that the shortest path is implicit once the destination node is brought inside the horizon. This algorithm inspired the contour expansion technique used in the BP_i algorithm, although the semantics are slightly different. For a description of Dijkstra's algorithm, see pages 204–208 in [Aho *et al.* 1983].

How This Thesis Fits In

We have seen how early work in robotics identified the important problem areas of task geometry, mechanics, and uncertainty, and that subsequent work has continued to explore these areas. This thesis attempts to combine these results and develop a unified approach to the analysis and planning of planar manipulation tasks. For example, one view of the thesis is that it extends the LMT formalism to include nonlinear task mechanics, as well as simultaneous consideration of translation and rotation for linear mechanics. Another view of the thesis is that it extends past work in mechanics analysis by producing general-purpose algorithms for analyzing arbitrary polygonal tasks; much of the previous work was accomplished using a hand analysis of a specific task geometry. Finally, the thesis contributes to the field of computational geometry by extending common representations of planar polyhedra to include a specific class of curved surfaces.

Conclusion

The previous sections have addressed an important problem facing manipulation research: the problem of constructing reliable actions to manipulate objects into desired positions. We have addressed a particular subclass of this problem that involves two planar polygonal objects, and defined five generic algorithms that may be applied to a variety of problems within this subclass. These algorithms have been implemented, and physical experiments have demonstrated that the algorithms synthesize reliable actions, provided that the appropriate physical assumptions are maintained. We then explored the implications of these results for future research, and concluded this Part by discussing the relationship between this thesis and the work that preceded it.

Reading up to this point has exposed you to all of the high-level results in this thesis. What remains are the details. In Part II, we will explore the nature of the sets that have been introduced, develop data structures to represent these sets, and present algorithms to construct the data structures. In order to improve readability, the hairy details are deferred to Part III, which addresses several topics that may interest readers who seek a deeper understanding of this work.

Part II
Models, Data Structures, and Algorithms

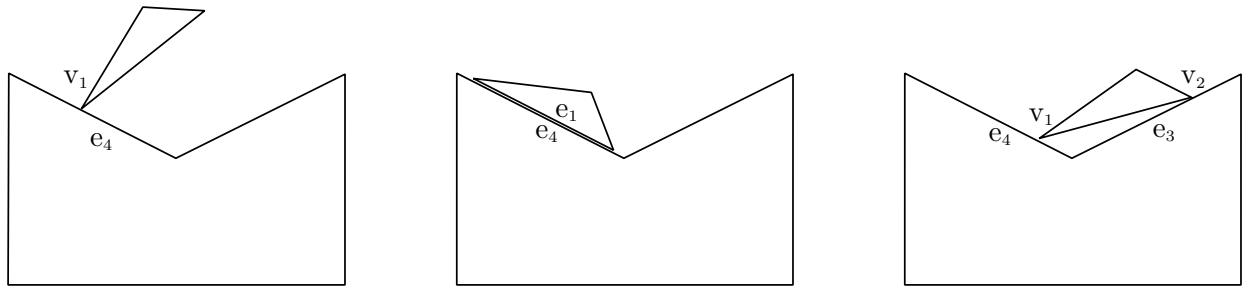
Kinematics

The Nature of Configuration-Space Obstacles

The kinematics of two polygons interacting in a plane can be completely described by identifying the reachable configurations of the two polygons. The set of reachable configurations is the union of the free configurations where the polygons do not touch, and the contact configurations where the polygons touch at one or more points. Both of these sets may be expressed in the (x, y, θ) space of relative polygon positions, or the *configuration space.* In the configuration space, the contact configurations correspond to the surface of the *configuration-space obstacle,* while the free configurations correspond to the points outside this surface.

The configuration-obstacle surface corresponds to the set of configurations where the polygons are in contact. Features on this surface correspond to topological classes of contact configurations, identified by the polygon features that are in contact. In order to describe these features, we will introduce a simple notation for describing contacts.

We denote a single contact by the pair of features that are in contact, always listing the moving-polygon feature first. For example, if vertex 1 of the moving-polygon is in contact with edge 4 of the fixed-polygon, then we denote the contact v_1e_4 (Figure 32). Multiple-contacts are separated by dashes, such as the v_1e_4-v_2e_3 contact shown in Figure 32(c). The order of the contact pairs



(a) (b) (c)

Figure 32: Example contacts. (a) v_1e_4. (b) e_1e_4. (c) v_1e_4-v_2e_3.

is not significant, so v_1e_4-v_2e_3 is equivalent to v_2e_3-v_1e_4. We will also refer to types of contacts, by dropping the subscripts that refer to specific polygon features. In the previous examples, the first contact is a ve contact, and the second is a ve-ve contact. Whether specific or generic, these descriptions denote the set of feature-pairs that are in contact, or the *contact set* for a given contact configuration.

There is a direct correspondence between contact sets and configuration-obstacle features. For each facet, edge, and vertex of the configuration obstacle, the contact configurations corresponding to the obstacle feature all share the same contact set. Each feature corresponds to a single contact set, but sometimes a contact set may correspond to more than one obstacle feature. For example, we shall see later that two disconnected obstacle facets may share the same contact set.

In the discussion that follows, we will develop an intuitive understanding of the class of configuration obstacles that can arise for arbitrary pairs of polygons. We will begin by considering simple contacts and convex polygons, and then examine the more complex contacts that can arise for non-convex polygons. In each case, we will focus on the relationship between the contact situation and the corresponding configuration-obstacle feature.

Vertex-Edge Contacts

The simplest contact is the vertex-edge (ve) contact. Figure 33 shows an example ve contact; what are the (x, y, θ) configurations where this contact is maintained? Here it is helpful to imagine that the vertex is connected to the edge by a revolute hinge that is free to slide along the edge, as shown in Figure 33(b). The configurations that maintain the vertex-edge contact are exactly the configurations that are reachable by this two-link system.

Since the hinge allows rotation and sliding, this system has two degrees of freedom. Thus the corresponding set of reachable configurations is a two-dimensional surface embedded in the three-dimensional (x, y, θ) configuration space. Neglecting motion limits, this is an infinite, ruled surface that stretches to infinity in the x and y dimensions and is defined for all θ values; we will refer to such surfaces as *c-surfaces.* The c-surface for our ve example is shown in Figure 34, and can be generated by sweeping a line along a sinusoidal curve.

If we choose a specific point on the line containing the slider, we can identify the set of configurations where the slider-hinge is coincident with the chosen point. Since we have constrained the position of the slider, the degrees of freedom of the system are reduced from two to one, and the corresponding set of reachable configurations becomes a one-dimensional curve embedded in the two-dimensional c-surface. This curve is a helix, and is shown in Figure 35(a). Similarly, if we choose a specific hinge orientation, the set of reachable configurations is also a one-dimensional curve (actually a straight line), shown in Figure 35(b).

These observations allow us to express the motion limits of the real ve contact, which arise due to the finite length of the edge and the limited orientations reachable by the contact. The edge endpoints give rise to curves where the vertex is superimposed with each endpoint; feasible ve contact configurations must lie between these curves. Similarly, the edges on either side of the vertex give rise to two lines on the c-surface; feasible configurations must lie between these lines. Since both constraints must be simultaneously satisfied, only the points within both bands are feasible ve contact configurations (Figure 36). If both polygons are convex, then this set of points comprises the configuration obstacle facet for this ve contact. If either polygon is non-convex, then some or all of these configurations may be unreachable, as we shall see later.

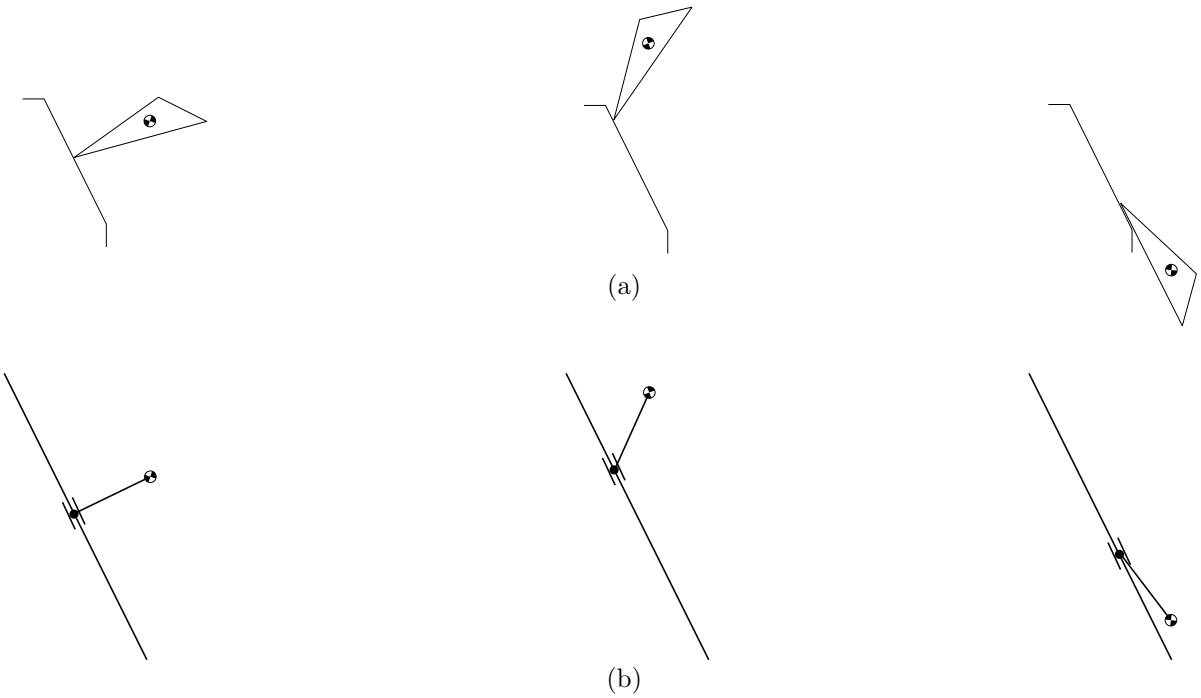


Figure 33: (a) A vertex-edge contact. (b) The corresponding abstract mechanism.

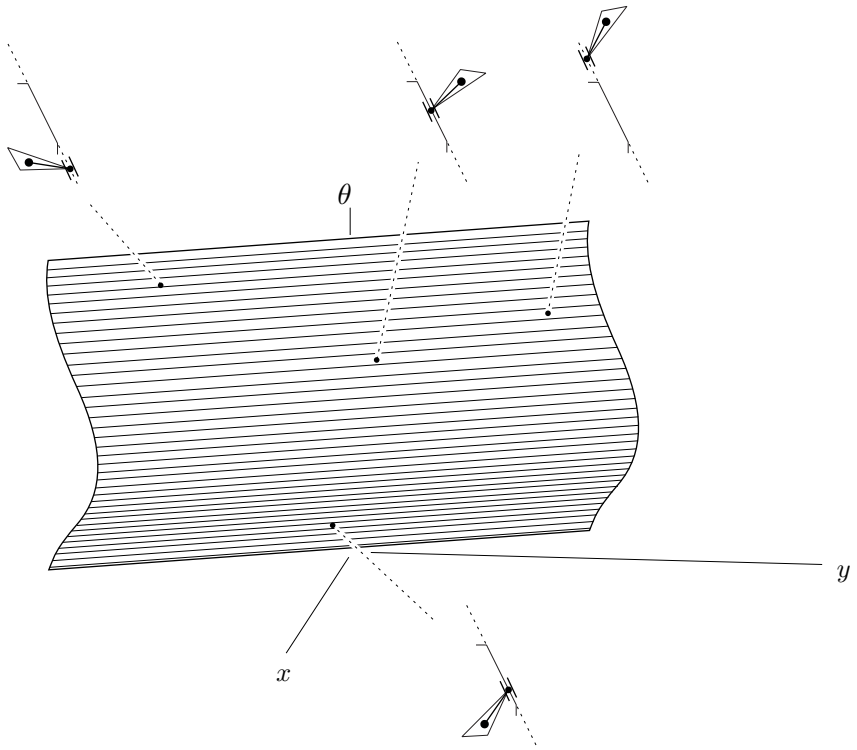
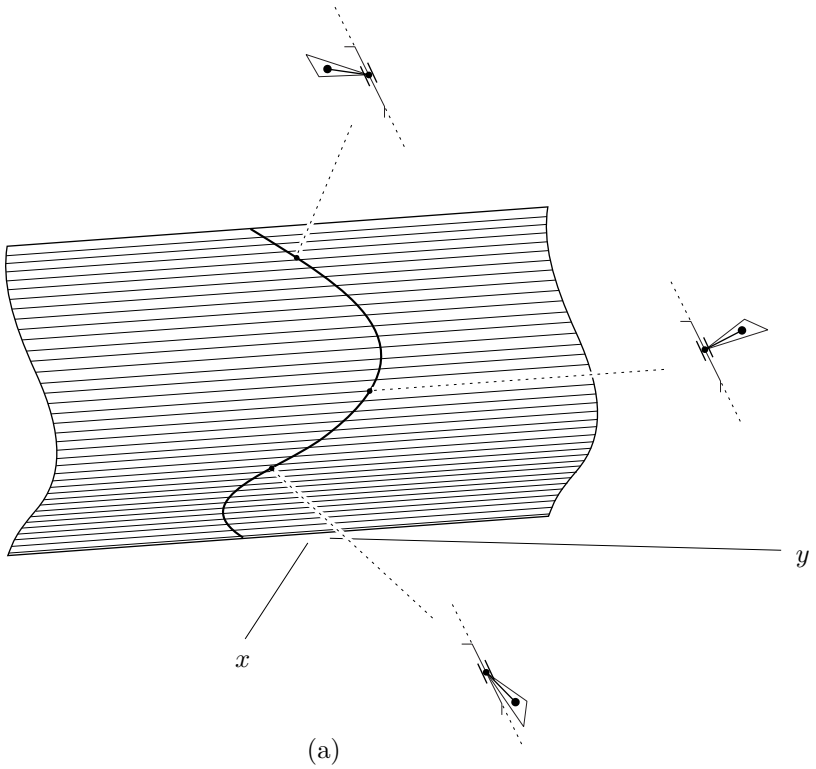
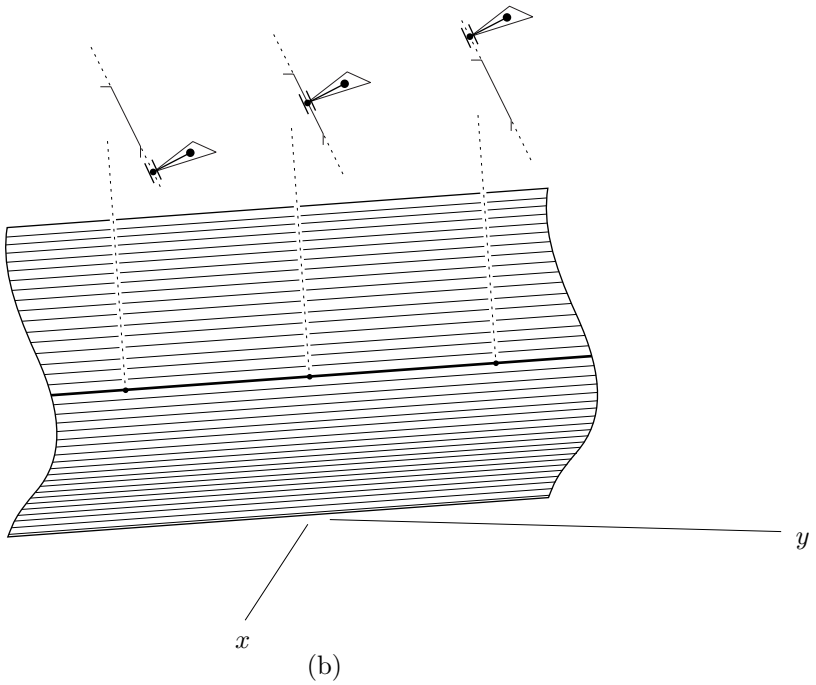


Figure 34: The set of reachable configurations for the abstract mechanism corresponding to a vertex-edge contact.



(a)



(b)

Figure 35: Fixing the slider and revolute joint of the abstract mechanism.

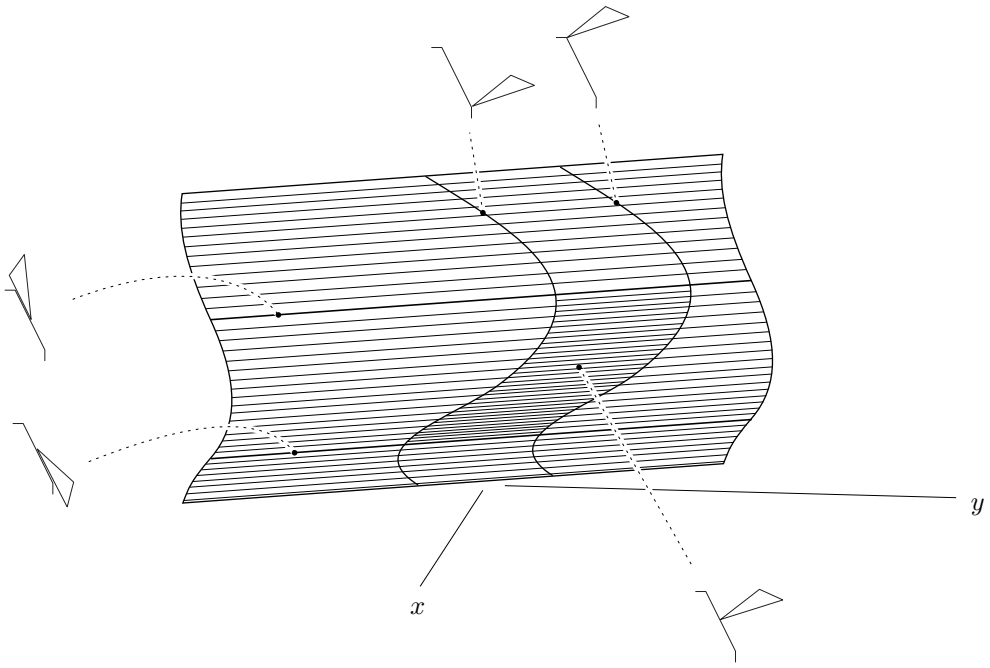
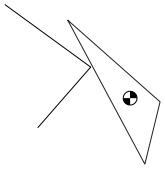


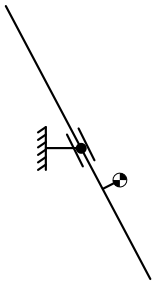
Figure 36: The locally-consistent subset of the unbounded c-surface for a vertex-edge contact.

Edge-Vertex Contacts

Figure 37 shows an example ev contact, and its analogous abstract mechanism. The set of reachable configurations for this abstract mechanism is the infinite helical surface shown in Figure 38; this surface is called a helicoid. As with the c-surface for the ve contact, this surface can be generated by sweeping a line; however in the ev case the orientation of the line varies continuously as it is swept along the generator curve. This surface has the same edge-endpoint and adjacent-edge motion limits as the ve case; these constraints are illustrated in Figure 39.



(a)



(b)

Figure 37: (a) An edge-vertex contact. (b) The corresponding abstract mechanism.

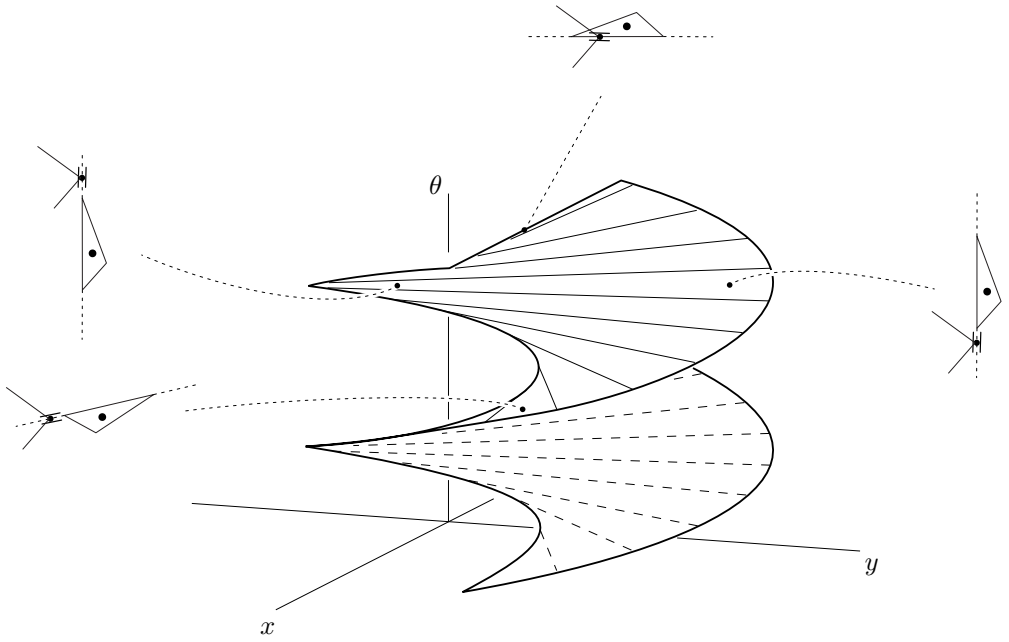


Figure 38: The set of reachable configurations for the abstract mechanism corresponding to an edge-vertex contact.

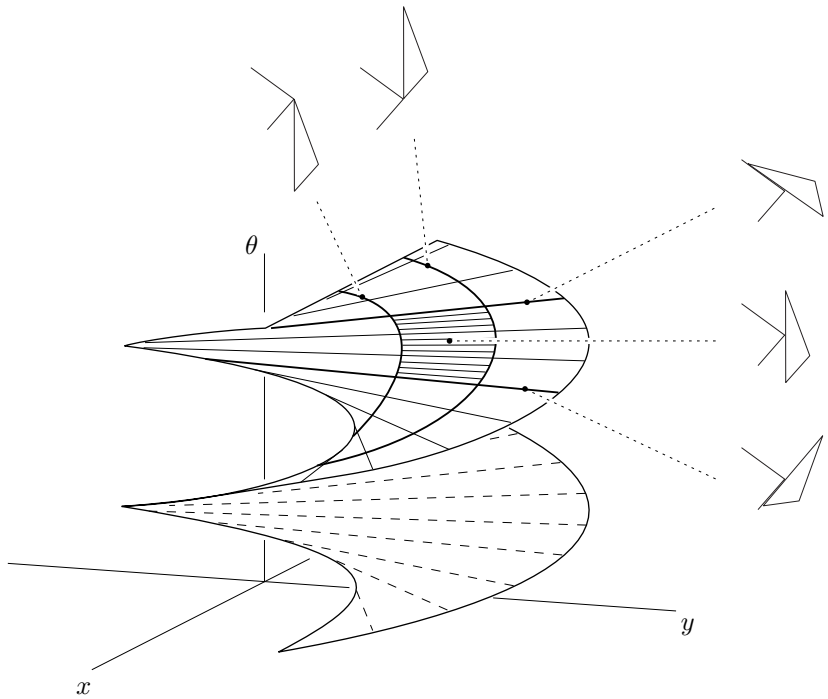


Figure 39: The locally-consistent subset of the unbounded c-surface for an edge-vertex contact.

Facet Neighbors

For configurations outside the limits imposed by the edge-endpoints or the adjacent-edges, a different contact condition applies. This new contact condition will have its own facet of reachable configurations. Thus the constraint curves form transitions between facets, and correspond to edges of the configuration obstacle. Figure 40 shows the neighboring facets for the previous ve contact example.

Since ve and ev facets account for all the possible contacts with two degrees of freedom, the ensemble of possible ve and ev facets forms the entire configuration obstacle surface. Every facet is surrounded by neighboring facets across all of its edges. Thus the configuration obstacle surface is closed. Note that since the θ-axis wraps around, surface points at $\theta = 0$ are connected to surface points at $\theta = 2\pi$. Figure 41 shows the configuration obstacle for a pair of convex polygons.

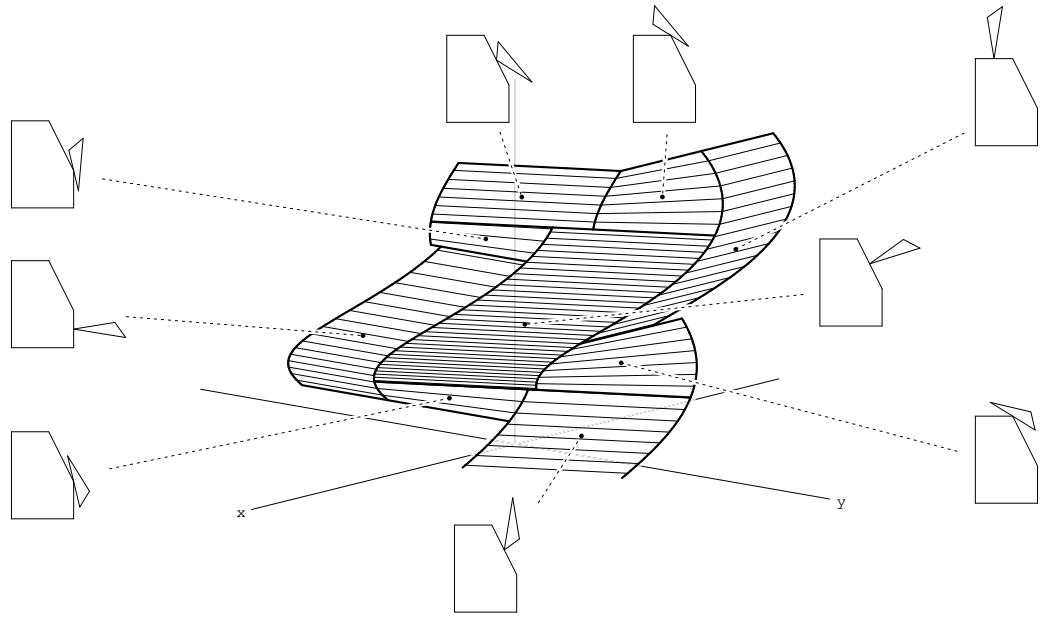


Figure 40: Facet neighbors.

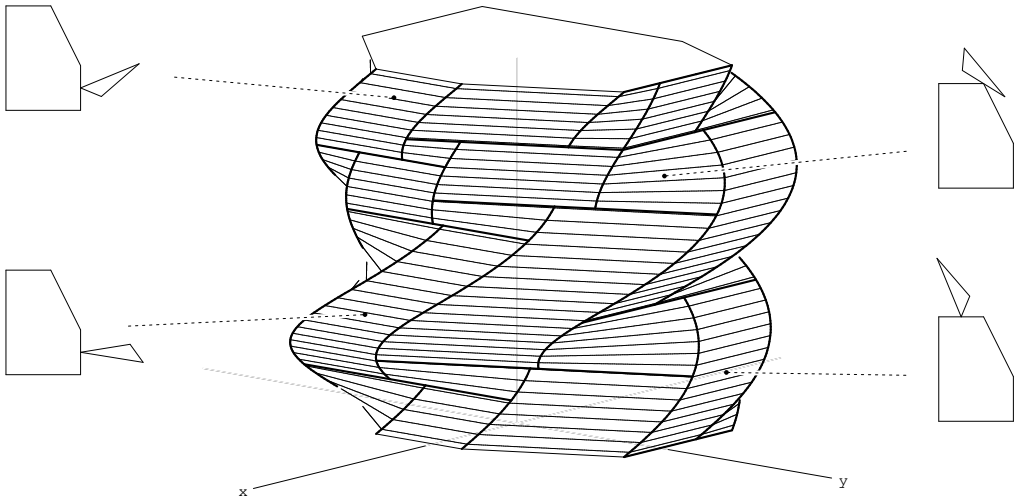


Figure 41: The configuration-space obstacle of two convex polygons.

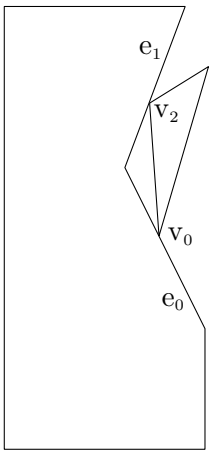
Multiple Contacts

So far we have seen all of the single-contact situations that are possible: ee, ev, ve, vv. These contacts give rise to facets and edges on the configuration obstacle surface; these are the only features that will appear if the moving and fixed polygons are both strictly convex. However, if either polygon is not strictly convex, then multiple-contact situations will also arise.

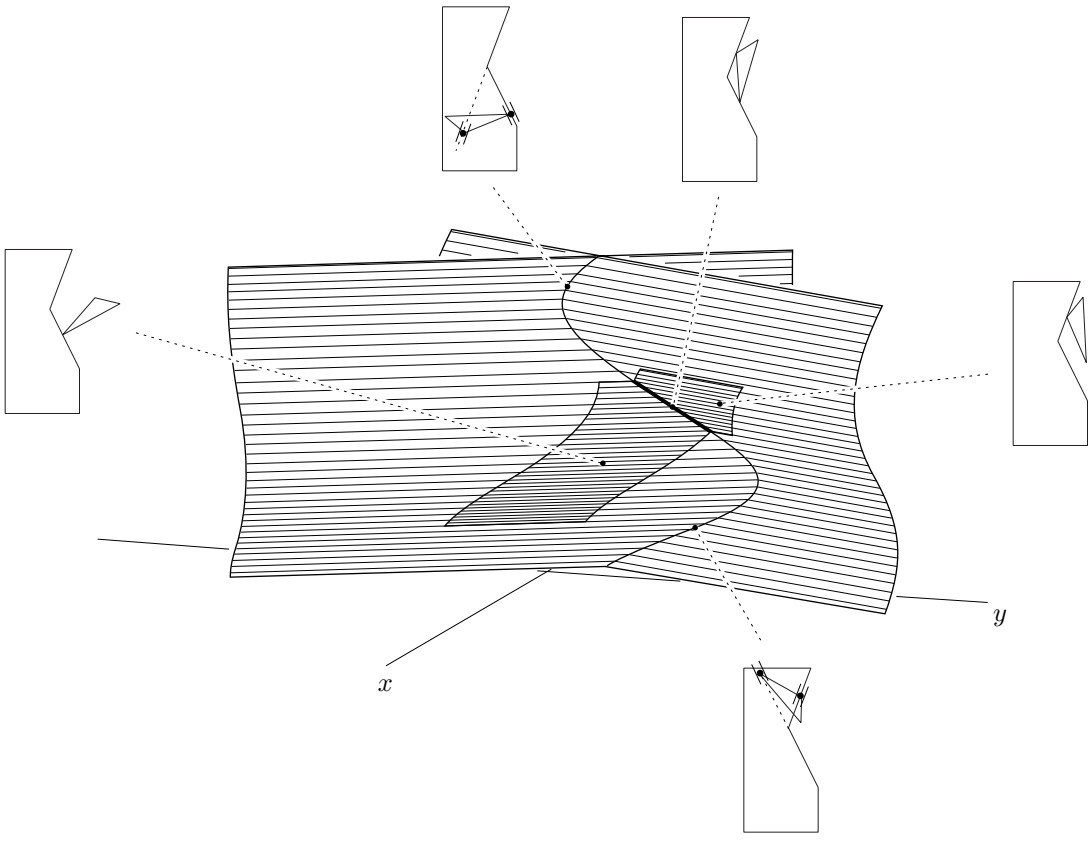
Consider the example shown in Figure 42(a). Here it is possible for the triangle vertex v_0 to contact the fixed edge e_0 at the same time that vertex v_2 is contacting edge e_1. Because of the presence of edge e_1, some of the v_0e_0 configurations cannot be reached without causing e_1 to intersect the triangle. Thus e_1 restricts the reachable v_0e_0 contact configurations; similarly, e_0 restricts the reachable v_2e_1 configurations. These observations suggest that the corresponding facets of the configuration obstacle will be reduced.

Figure 42(b) illustrates this situation in the configuration space. The infinite c-surfaces containing the v_0e_0 and v_2e_1 facets are shown, as well as the facet edges arising from the local constraints. The c-surfaces intersect, defining the set of configurations satisfying both contacts. If we visualize the v_0e_0-v_2e_1 contact as an abstract mechanism with hinge-sliders at each ve contact, then the curve determined by the c-surface intersection corresponds to the set of reachable configurations for the mechanism. The set of reachable configurations for the real v_0e_0-v_2e_1 contact is a subset of this curve, delineated by the local constraints of the v_0e_0 and v_2e_1 facets.

The curve defining the intersection of two ve c-surfaces is always well-defined, as long as the lines containing the fixed-polygon edges are not parallel. This is a direct consequence of the fact that non-parallel lines lying in a plane always have a point of intersection. This implies that the abstract mechanism corresponding to any ve-ve contact always has a set of reachable configurations, as long as the fixed edges are not parallel. However, the real ve-ve contact may have no reachable configurations, since there may be no points on the intersection curve that simultaneously satisfy the local constraints of each facet (Figure 43).



(a)



(b)

Figure 42: (a) A multiple-contact condition. (b) The corresponding situation in the configuration space. The intersection of the c-surfaces corresponds to the reachable configurations for the v_0e_0-v_2e_1 abstract mechanism, and the bold subset of this intersection corresponds to the reachable configurations for the real v_0e_0-v_2e_1 contact.

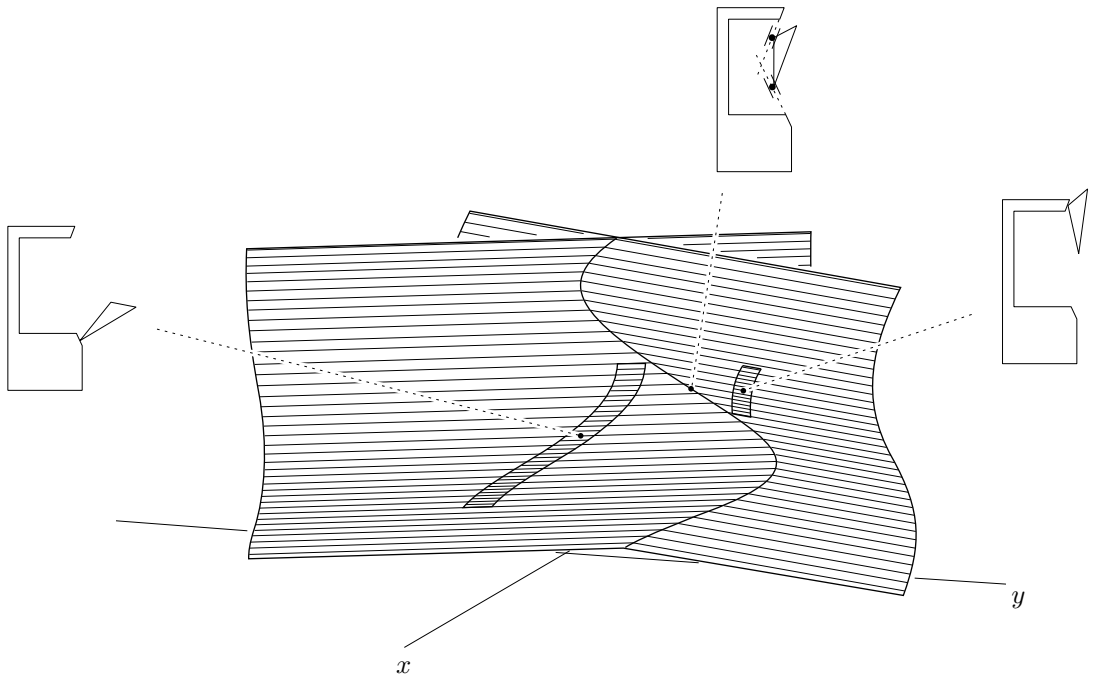


Figure 43: A multiple-contact condition that has no locally-consistent configurations.

What if the fixed edges are parallel? In this case the ve-ve abstract mechanism will have reachable configurations exactly when the distance between the vertices is greater than or equal to the distance between the lines containing the edges. When this condition is met, the ve c-surfaces will intersect in one or two lines (Figure 44). These lines form the set of reachable configurations for the ve-ve abstract mechanism; it is evident from the labels in the figure that these configurations may slide laterally, but have no rotational freedom. This is consistent with the fact that the lines in the configuration space lie in a constant-θ horizontal plane. As in the case of non-parallel edges, the existence of reachable configurations for the abstract mechanism does not imply the existence of reachable configurations for the real ve-ve contact (Figure 45).

Up to now we have ignored the distinction between parallel and antiparallel fixed edges. Since edges are in fact directed lines, parallel and antiparallel cases are not topologically the same. When the fixed edges are parallel, the moving-polygon is free to rotate either clockwise or counter-clockwise, maintaining one of the two ve contacts for each rotation direction (Figure 46(a)). When the fixed edges are antiparallel, then rotation is only possible in one direction, and either ve contact may be maintained for that rotation direction (Figure 46(b)).

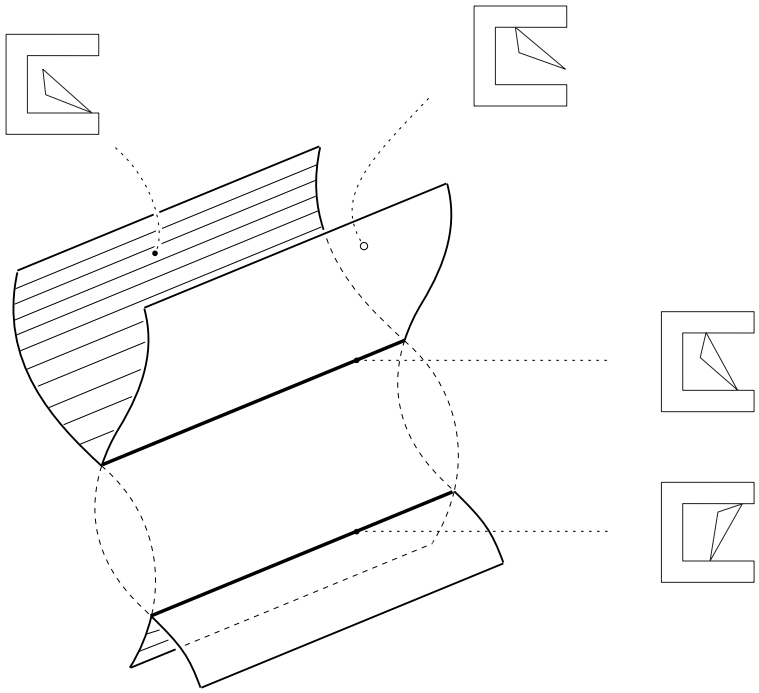
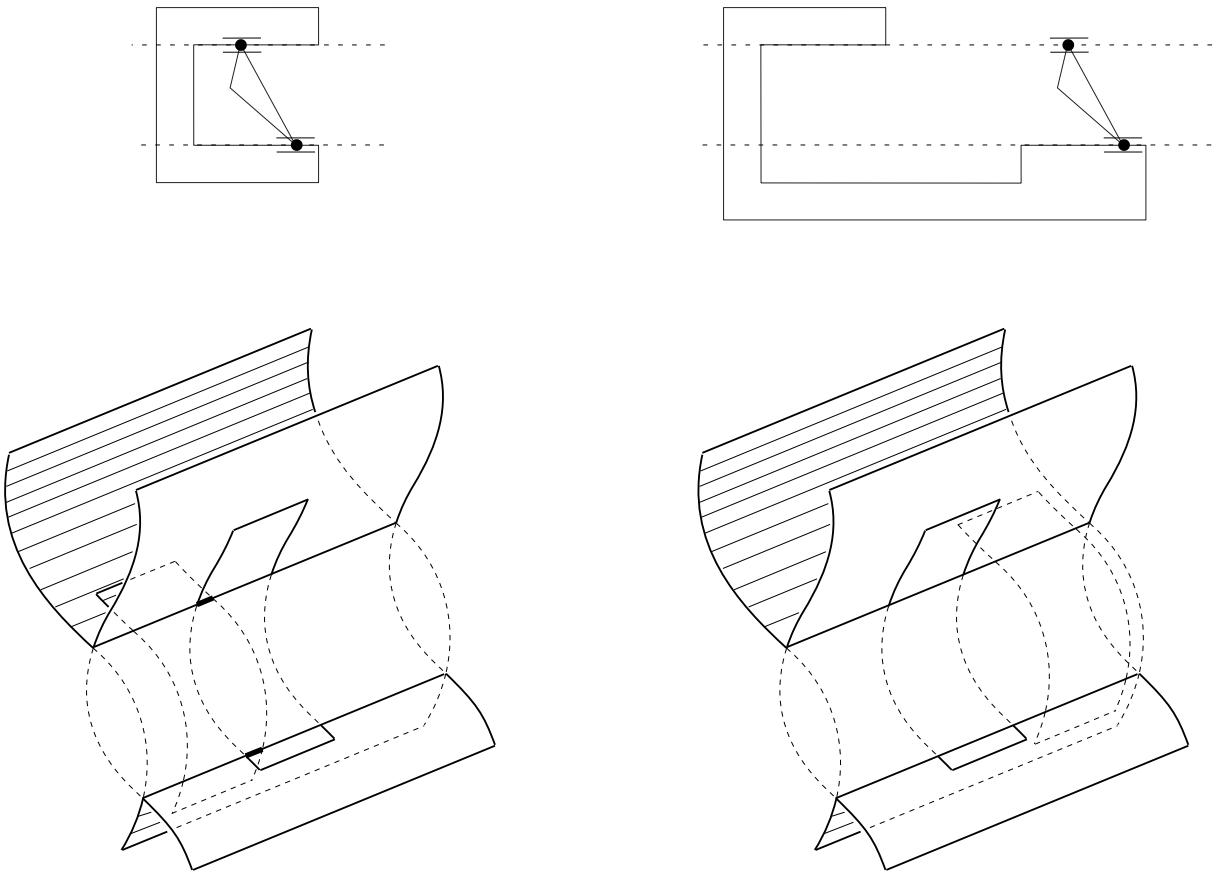
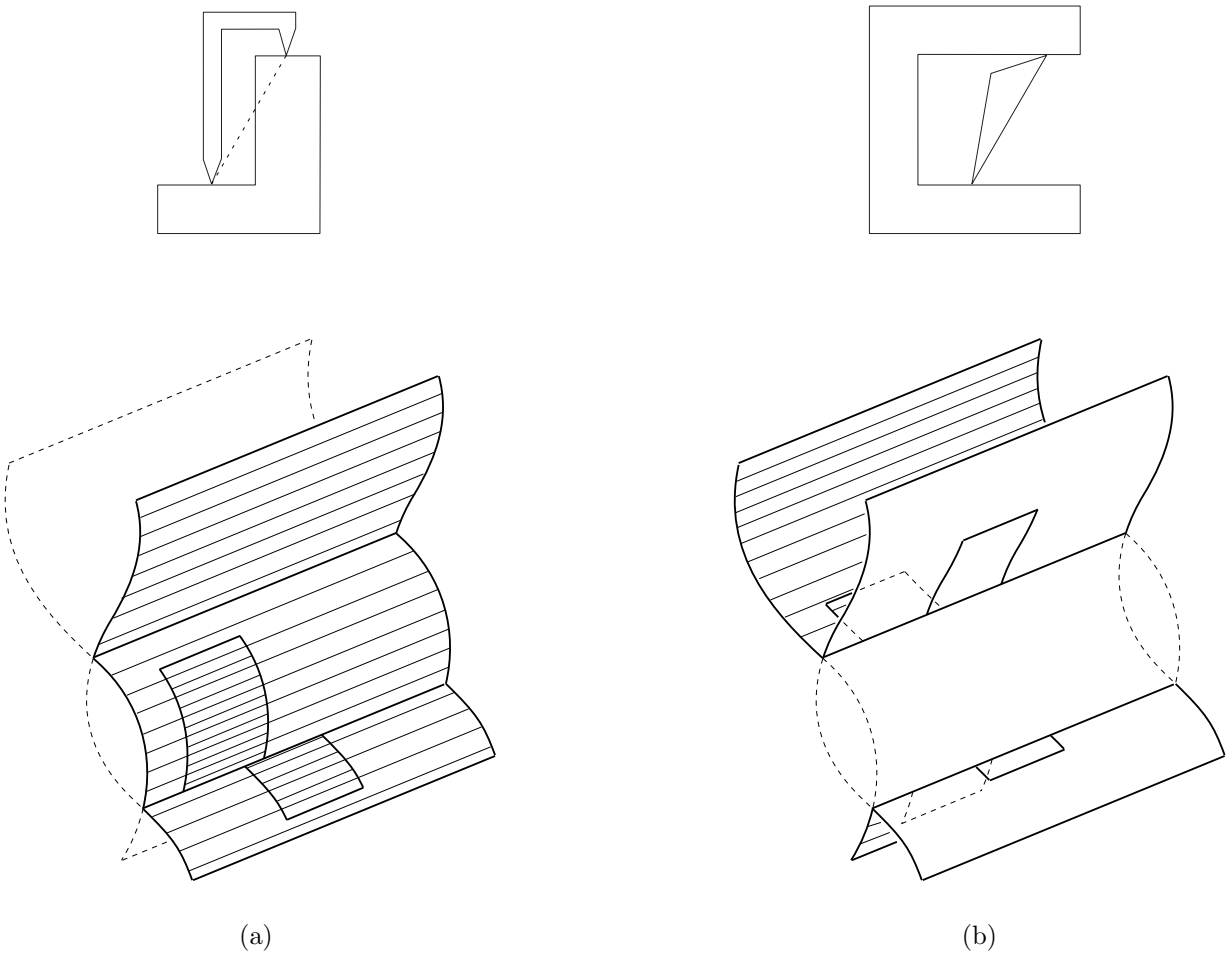


Figure 44: The intersection of c-surfaces for a ve-ve contact with parallel fixed edges.



(a) (b)

Figure 45: The effect of local-consistency constraints on multiple-contacts with parallel fixed edges. (a) and (b) share the same abstract mechanism, but only (a) contains locally-consistent contact configurations.

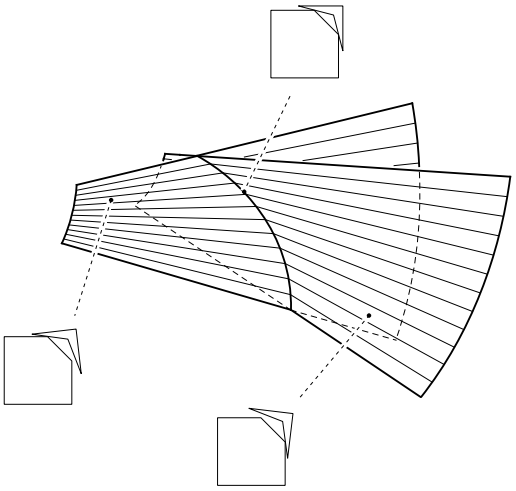


(a) (b)

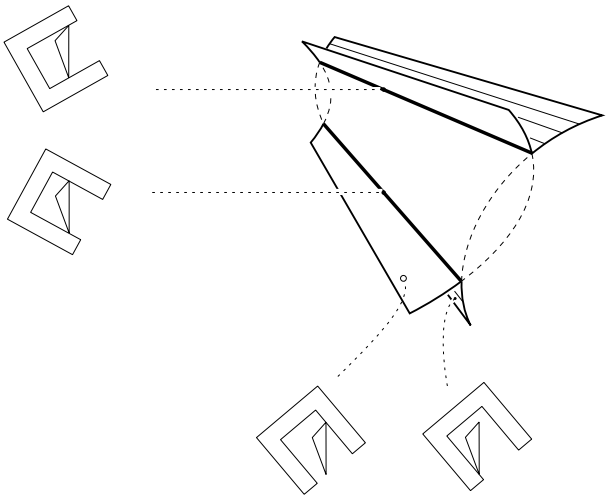
Figure 46: The distinction between parallel and antiparallel edges.

The previous paragraphs have explored the nature of ve-ve contacts; we have considered cases where the fixed-edges are intersecting, and where they are aligned (i.e, either parallel or antiparallel). These cases produced obstacle edges that were either functions of θ, or lines confined to a constant-θ plane. Analagous cases are possible for ev-ev and ve-ev contacts; Figures 47 and 48 show some examples.

Since only ve or ev facets are possible, the only two-contact obstacle edges that are possible are the 2-combinations of these facets: ve-ve, ev-ev, and ve-ev. For each of these, there are cases where the edges involved are aligned, or non-aligned. Since these edges arise because of a conflict between single contacts, we will refer to them as *conflict edges.* Similarly, we will refer to the vv or ee edges that arise from the local edge-endpoint or adjacent-edge constraints as *local edges.* The local and conflict edges exhaust the set of possible configuration obstacle edges, except for a special class of non-generic edges that are presented in the next section.



(a)



(b)

Figure 47: Example ev-ev contacts. (a) Non-aligned. (b) Aligned.

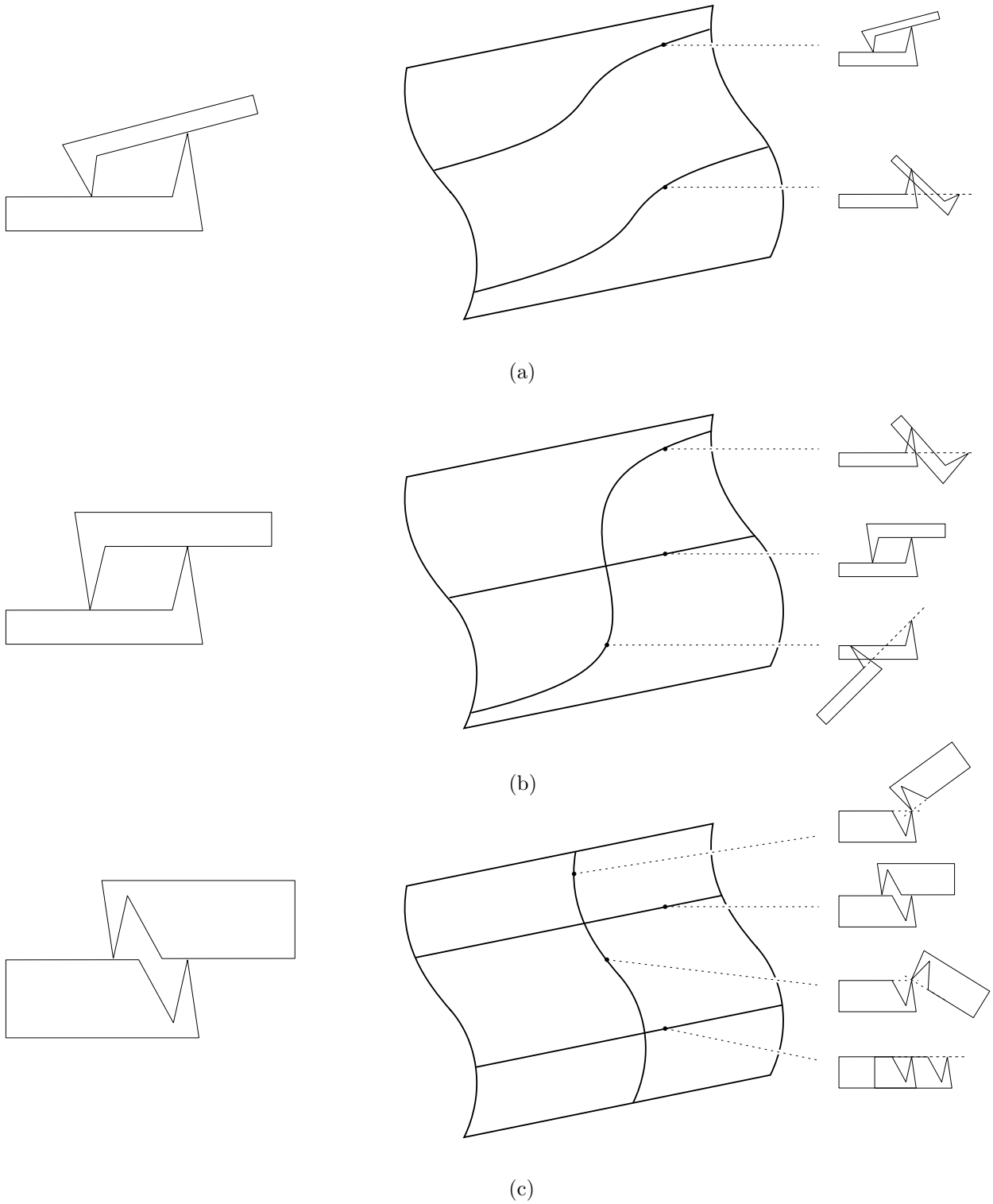


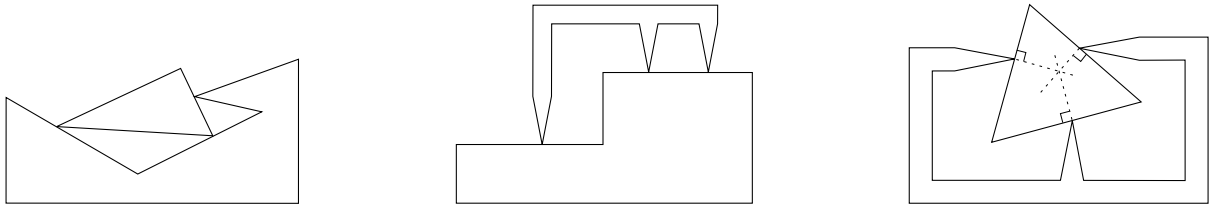
Figure 48: Example ve-ev contacts, with the ev surfaces omitted for clarity. (a) General situations produce two disjoint c-surface intersection curves, each of which is a function of θ that approaches infinity at certain critical values. (b) When the vertex-edge distances are equal, the c-surfaces intersect in two curves, one of which is a function of θ, and one of which is a constant-θ line. (c) When both vertex-edges distances are zero, the c-surface intersection produces a helix and two constant-θ lines.

Non-Generic Contacts

If the moving-polygon touches the fixed-polygon at more than two contact points, then the moving-polygon cannot undergo a finite motion while maintaining the contacts unless all of the contact normals lie on parallel lines. If this condition is met, the moving-polygon may translate while maintaining all of the contacts. This implies that the only configuration-obstacle edges that are possible with more than two contacts are constant-θ edges. Example configurations with more than two contacts are shown in Figure 49.

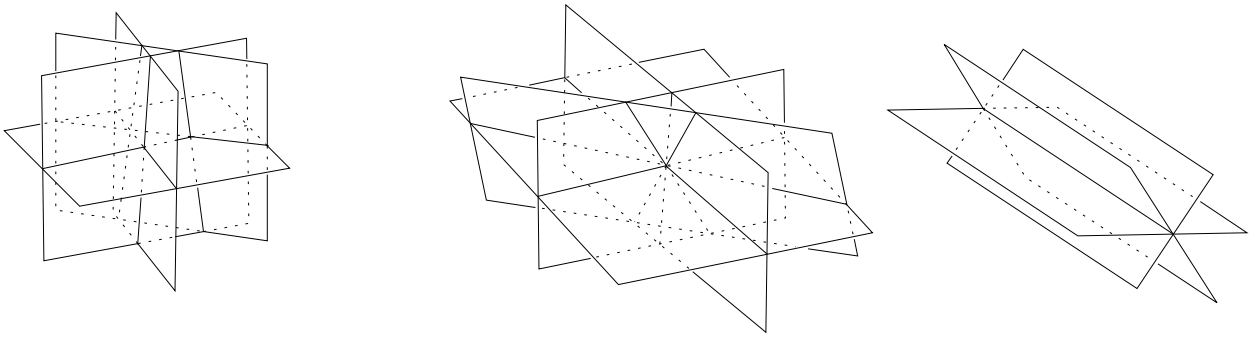
Configuration-obstacle edges with more than two contacts arise only when geometric coincidences are present; the polygon features involved must be separated by exactly the right inter-feature distances. These contact situations are *non-generic*.

When constructing and manipulating configuration obstacle surfaces, it is useful to distinguish between generic and non-generic situations. To illustrate the notion of generic topology, let's consider the simple example of planes intersecting in three-dimensional Euclidean space. In general, two planes intersect at a line, three planes intersect at a point, and four planes have no common intersection. This is the topology of generic plane intersections. However, it is possible to choose singular configurations of planes that have very different intersection topologies; for example, an arbitrary number of planes may intersect at a given line or point. Such configurations are non-generic. Figure 50 shows examples of generic and non-generic plane intersections.



(a) (b) (c)

Figure 49: Contact situations with more than two contact points. (a) A contact configuration with no degrees of freedom. (b) A contact configuration with one translational degree of freedom. (c) A contact configuration with one rotational degree of freedom; only infinitesimal motion is possible.



(a) (b)

Figure 50: (a) Generic intersection of four planes. (b) Non-generic plane intersections.

The intersections of c-surfaces are somewhat more complicated, but still have identifiable generic and non-generic topologies. In general, two c-surfaces intersect at up to two curves, three c-surfaces intersect at up to four points, and four c-surfaces have no common intersection. However, non-generic intersection topologies are possible, and frequently arise in practice. Figure 51 shows several non-generic contact situations, and their corresponding c-surface intersections. Notice that some unusual intersection topologies arise; in (c–f), several c-surfaces intersect at a single line, while in (a),(b), (g), and (h), several surfaces intersect at a common point. The resulting topology of the configuration obstacle surface is also unusual. Cases (a–d) have normal surface topology, but (e) produces an isolated obstacle edge in the middle of a facet, (f) produces a spur edge that has no neighboring facets, (g) produces an isolated vertex on an obstacle facet, and (h) produces an isolated obstacle vertex with no adjacent edges or facets. The possiblity of spur edges and isolated vertices implies that general configuration-obstacle surfaces are not two-dimensional everywhere; these singular cases place extra demands on algorithms that seek to produce complete representations of configuration obstacles. In fact, the *CO* algorithm reported in this thesis does not construct the features shown in (e–h), as we shall see in later sections.

The constant-θ edges that can arise in non-generic situations form the last type of edge that is possible for configuration obstacles arising from a pair of polygons. Unlike the constant-θ obstacle edges due to ee, ve-ve, ev-ev, or ve-ev contacts, these non-generic edges may have arbitrarily many contacts that are simultaneously satisfied (Figure 52). Because these non-generic obstacle edges may have arbitrarily complex contacts, we cannot denote this class of edges by a particular contact specification. Instead, we will refer to these non-generic edges as *multiple-class-0 edges,* or simply *mc0-edges.* This terminology alludes to the fact that these non-generic obstacle edges may be viewed as the superposition of several different constant-θ edges, which we will later refer to as class-0 edges when we describe configuration obstacle features mathematically. Note that this term is more specific than the term "non-generic edge," since it is possible for non-generic edges to arise for ordinary two-contact conditions (Figure 51(e)).

Multiple-class-0 edges may give rise to obstacle edges that are physically significant, but are not topologically significant features of the obstacle surface. Figure 53 shows an example. In this three-contact multiple-class-0 edge, the outermost ve contacts are maintained over a wider interval of configurations than the innermost ve contact. On the configuration-space obstacle surface, the mc0-edge corresponding to this non-generic contact is bordered on either side by ve-ve edges that are collinear with the mc0-edge, and the adjacent facets are the same for all three edges. The reason that this edge has been set apart is not apparent from the obstacle surface topology, but depends on the contact information associated with points on the surface.

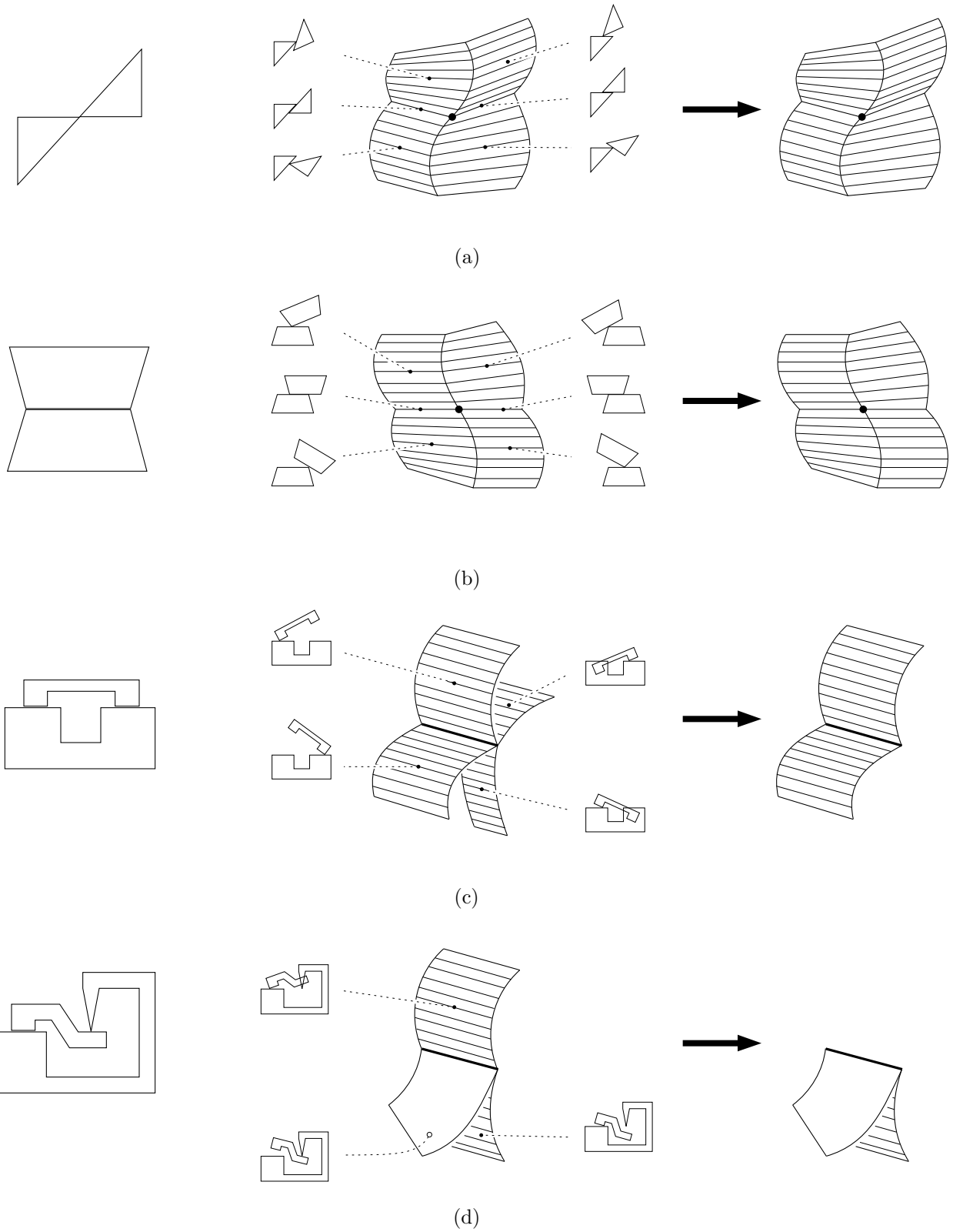
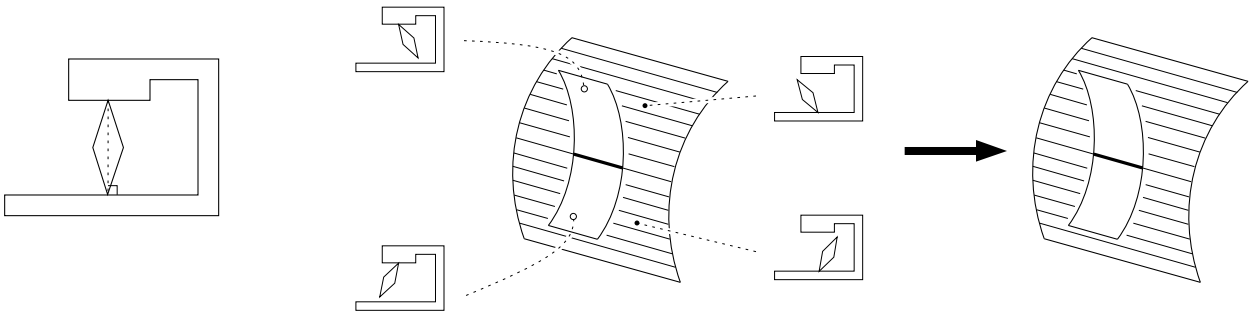
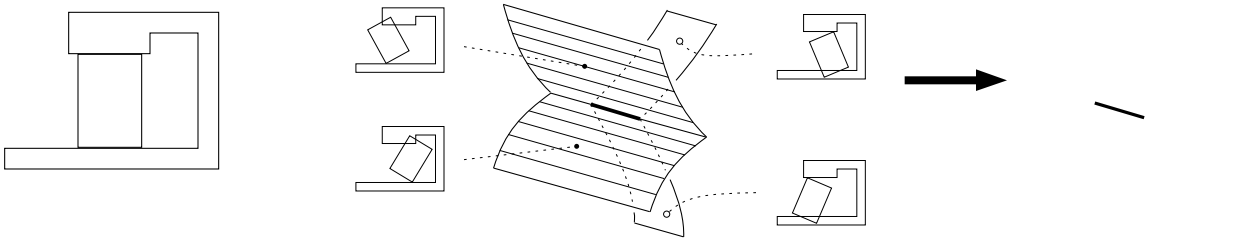


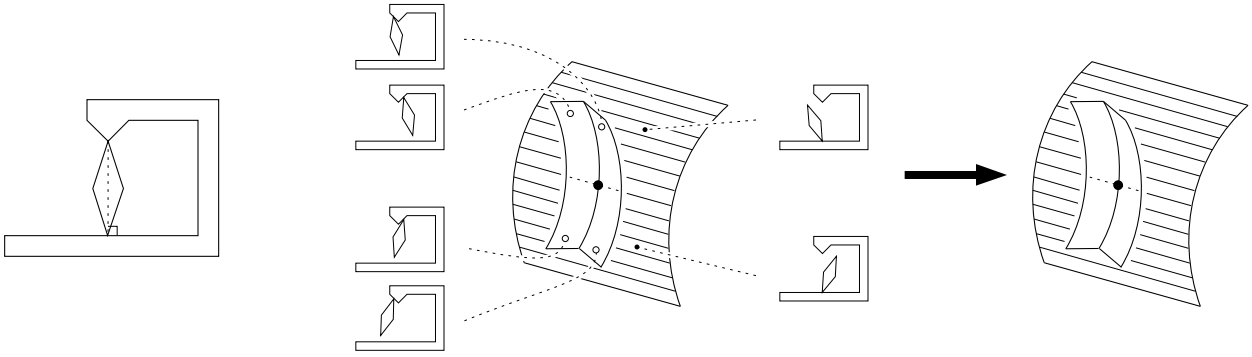
Figure 51: Non-generic contact situations. Each case shows a non-generic contact, its associated non-generic c-surface intersection, and the final set of reachable configurations.



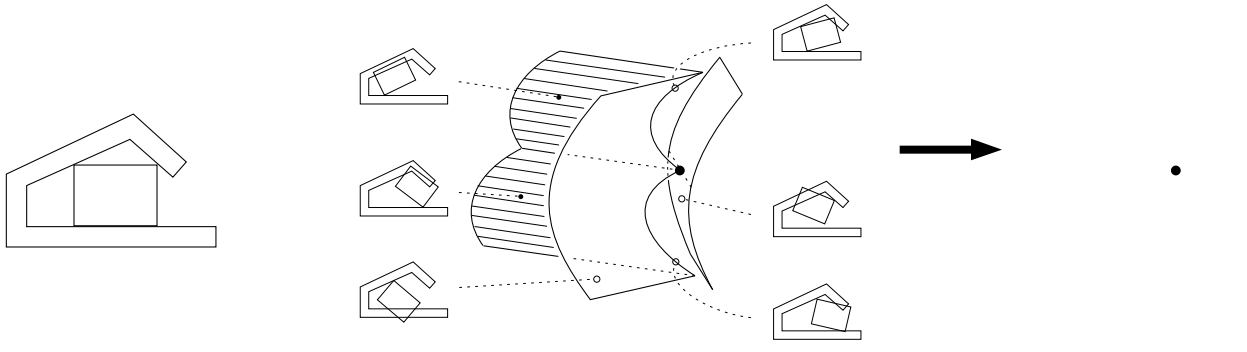
(e)



(f)



(g)



(h)

Figure 51 (continued): Non-generic contact situations. Each case shows a non-generic contact, its associated non-generic c-surface intersection, and the final set of reachable configurations.

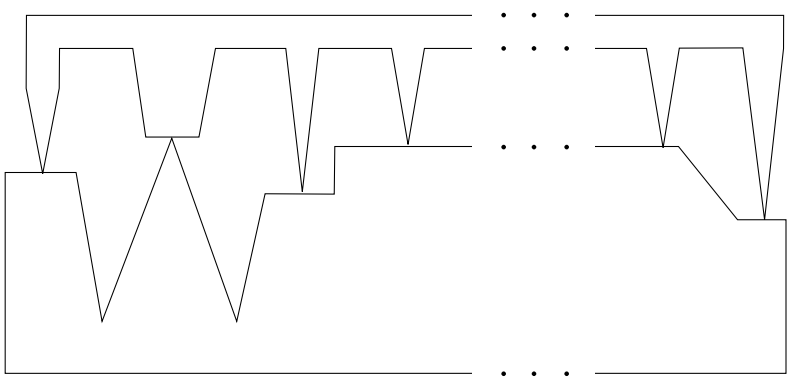


Figure 52: A complex non-generic contact.

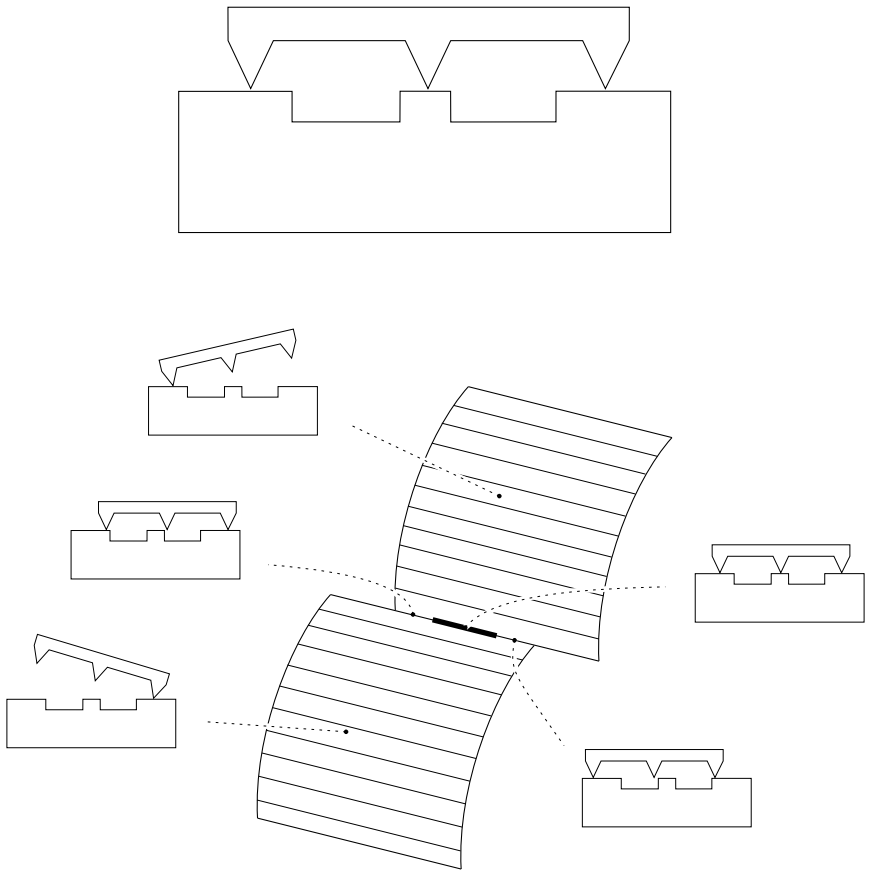


Figure 53: A multiple-class-0 edge that is physically significant, but does not correspond to a topological event on the configuration obstacle surface.

Why Non-Generic Situations Are Important

Non-generic contact conditions exist only when a geometric coincidence is present. While these coincidences may at first seem rare and implausible, many of them frequently arise in practical situations. For example, the non-generic contacts (a) and (b) in Figure 51 require equal angles or equal-length edges; these coincidences will arise multiple times whenever the moving-polygon and the fixed-polygon are the same shape. The configuration shown in (c) arises when a polygon has two collinear edges; physical objects with this characteristic can easily be constructed by cutting a straight edge, and then cutting a notch in the edge. When the polygons describe parts that are designed to fit together tightly, the contact situation shown in (f) will arise, unless detailed tolerance information is included in the input description of the polygon shapes.

Many past algorithms for constructing configuration-obstacle surfaces have assumed that non-generic situations do not arise. To handle the non-generic situations that do arise, the algorithms perturb the input polygons slightly until the situation becomes generic. The result of this perturbation is a configuration-space obstacle that contains no non-generic surface intersections — this obstacle is topologically different from the true configuration obstacle. Since the change in shape is small, this perturbation does not have a deleterious effect on path-planning, which has been the principal application for which these obstacles have been constructed.

However, this change is inappropriate for a physical analysis of interacting polygonal objects, because the modifications to the configuration-obstacle lead to incomplete descriptions of the contact set for some configurations. Since interaction forces occur at contacts, this may lead to an inaccurate description of the system's physical behavior. For example, material differences may cause the middle ve contact in Figure 53 to have a higher coefficient of friction than the outer ve contacts, causing the behavior of configurations on the corresponding mc0-edge to differ from the behavior of the adjacent ve-ve configurations. Failure to detect this non-generic contact could lead to erroneous physical predictions.

General Properties of Configuration-Space Obstacles

So far we have seen all of the features that can appear on the surface of a configuration obstacle. We have seen that the surface of a configuration obstacle is comprised of an ensemble of ve and ev facets, each of which is a ruled surface. The edges connecting these facets correspond to more constrained contact conditions that have only one degree of freedom; some of these edges may be identified using purely local information (vv and ee edges), while others result from non-local interactions of polygon features (conflict and mc0-edges). At the intersections of obstacle edges are vertices that correspond to contact conditions that have zero degrees of freedom. Each of these features can be described by a set of parameterized equations; these equations will be developed in the next section.

In addition to identifying the individual features that comprise a configuration obstacle surface, we can also study the overall surface properties of configuration obstacles. Some of these overall properties are listed below:

- *The total surface is bounded, closed, possibly nested, not necessarily connected, and of arbitrary positive genus.* Since the configuration obstacle is an ensemble of a finite number of bounded facets, the obstacle surface is bounded. Further, since the obstacle surface separates the free and impossible configurations, it must be closed. However, the total surface may be comprised of multiple nested shells, as shown in Figure 54. Further, the surface may contain disconnected non-nested shells, as shown in Figure 55. Finally, all configuration obstacles have genus 1 or higher because the (x, y, θ) space in which they are embedded imposes the topology of the torus. The genus may be arbitrarily large, because the geometry of the interacting polygons may give rise to an arbitrary number of handles on the obstacle surface; Figure 56 shows a geometric situation that produces a configuration-obstacle handle.
- *Obstacle facets may be split, or have holes.* As mentioned previously, it is possible for a single contact set to correspond to multiple configuration-obstacle features. This occurs when the locally-consistent subset of the contact c-surface is split by non-local feature interactions; Figure 57 shows how this may occur. Notice that both the facet and the conflict edge are split into disconnected parts. By a suitable choice of geometry, a facet may be split into arbitrarily many disconnected parts (Figure 58). Further, holes may be pierced in the facet that do not touch the local constraint edges; Figure 59 shows how there may be an arbitrary number of these holes in a single facet. Note that these holes are not holes in the obstacle surface, which is always closed. Rather, the holes result from the intersection of the facet with some other portion of the obstacle surface.
- *Not all facets are reachable.* Non-local feature interactions may make some facets completely unreachable (Figure 60).
- *Configuration-obstacle surfaces are not manifolds.* The term "manifold" has several interpretations in the field of algebra and computational geometry; we will consider two of these interpretations here.

  In the first interpretation, manifolds are defined to be continuously differentiable surfaces; a surface is a manifold if all of its derivatives are defined at every point on the surface. Under this definition, all configuration obstacles are clearly not manifolds, because the sharp edges between facets represent discontinuities in the surface orientation.

  The second interpretation of manifolds is more commonly used in computational geometry, and is concerned with the pure topology of the surface. Under this interpretation, a surface is a manifold if the neighborhood of all points on the surface is homeomorphic to a disc. In other words, the neighborhood surrounding any point of a manifold surface may be stretched and deformed into a disc without tearing the surface. Most configuration obstacles satisfy this definition, but some non-generic situations give rise to configuration obstacles that are not manifolds. For example, the spur edge illustrated in Figure 51(f) violates this manifold condition.
- *Duality relationships abound.* Vertex-edge and edge-vertex contacts are duals, which in turn gives rise to a variety of other dual relationships. For example, if the moving-polygon and fixed-polygon are interchanged, then the corresponding configuration-space obstacle has the same topology as the original obstacle, except the ve facets become ev facets, and *vice versa.*

This section has examined the qualitative nature of configuration-space obstacles and the features that comprise them. The next section will develop representations for these obstacles.

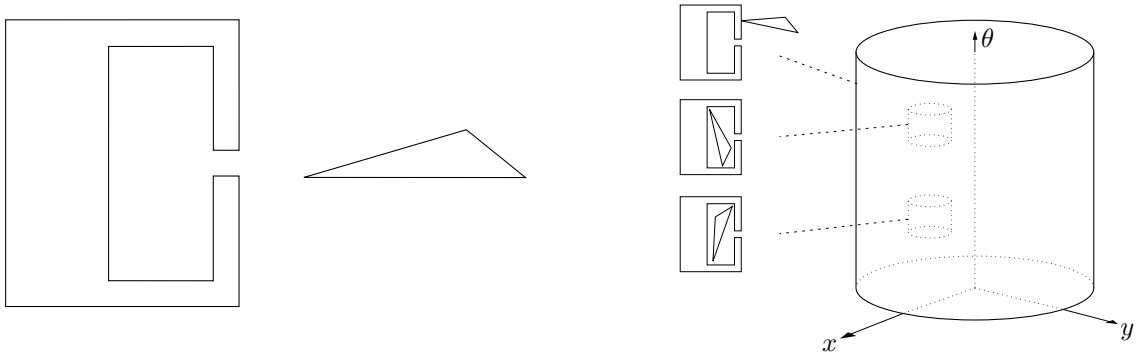


Figure 54: A situation giving rise to nested shells in a configuration-space obstacle. In Figures 54–56, a schematic representation of the obstacle is shown to clarify its topology.

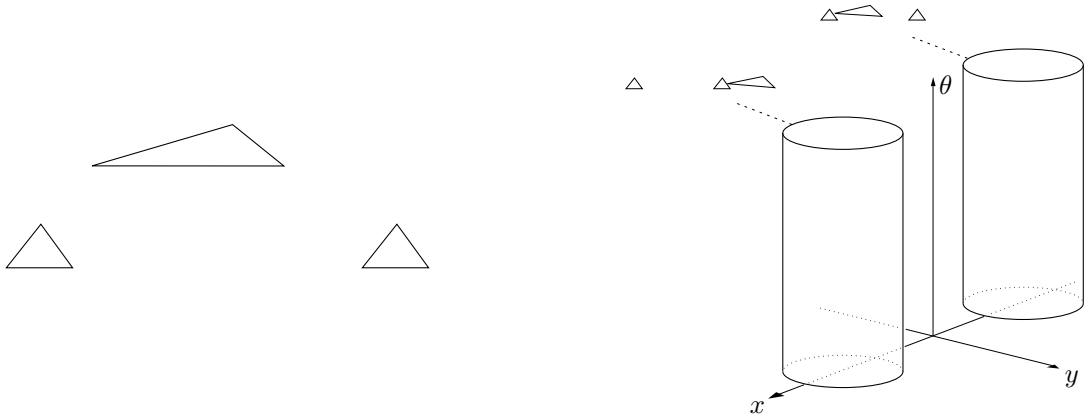


Figure 55: A situation giving rise to disjoint configuration-obstacles.

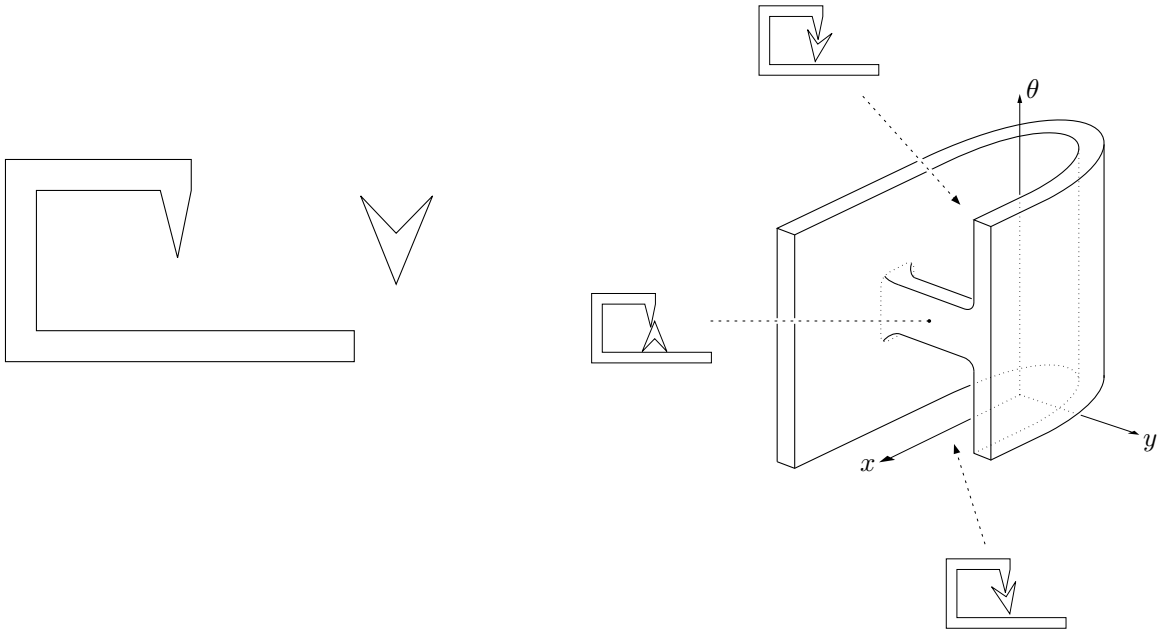


Figure 56: A situation giving rise to a handle on a configuration-space obstacle.

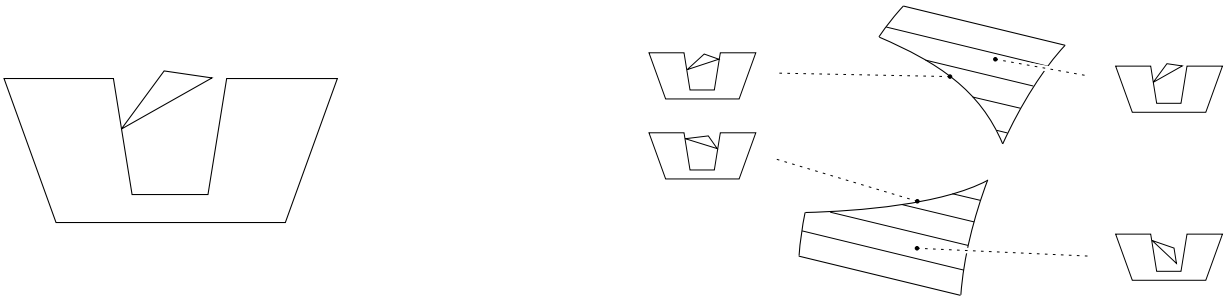


Figure 57: A facet and conflict edge that are split.

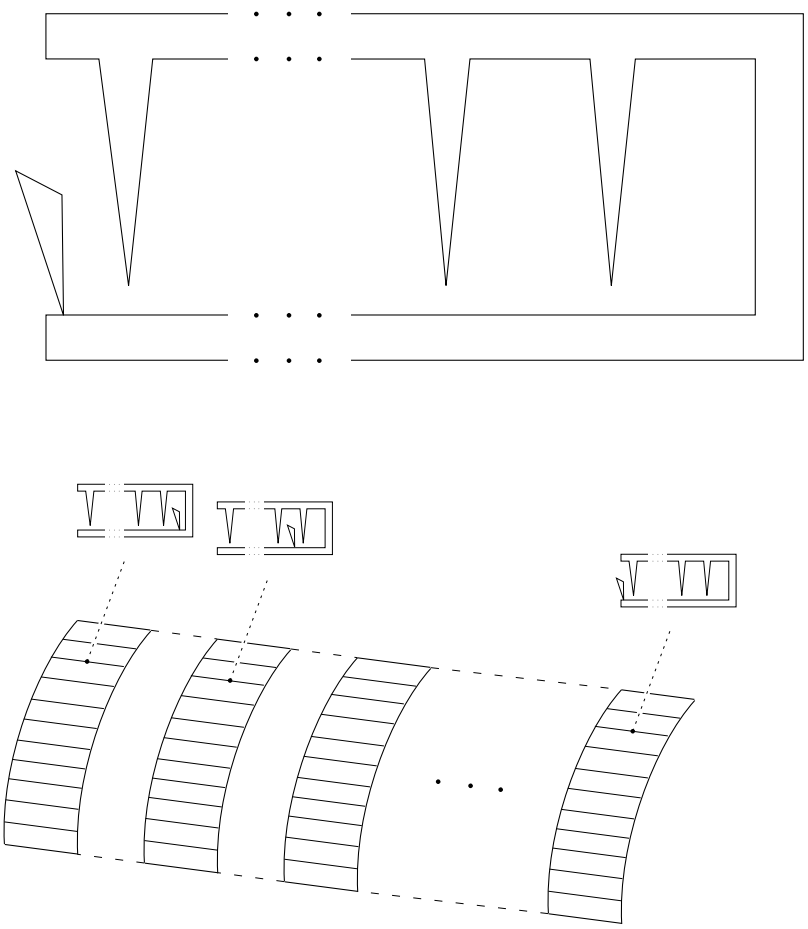


Figure 58: A facet may be split into arbitrarily many components.

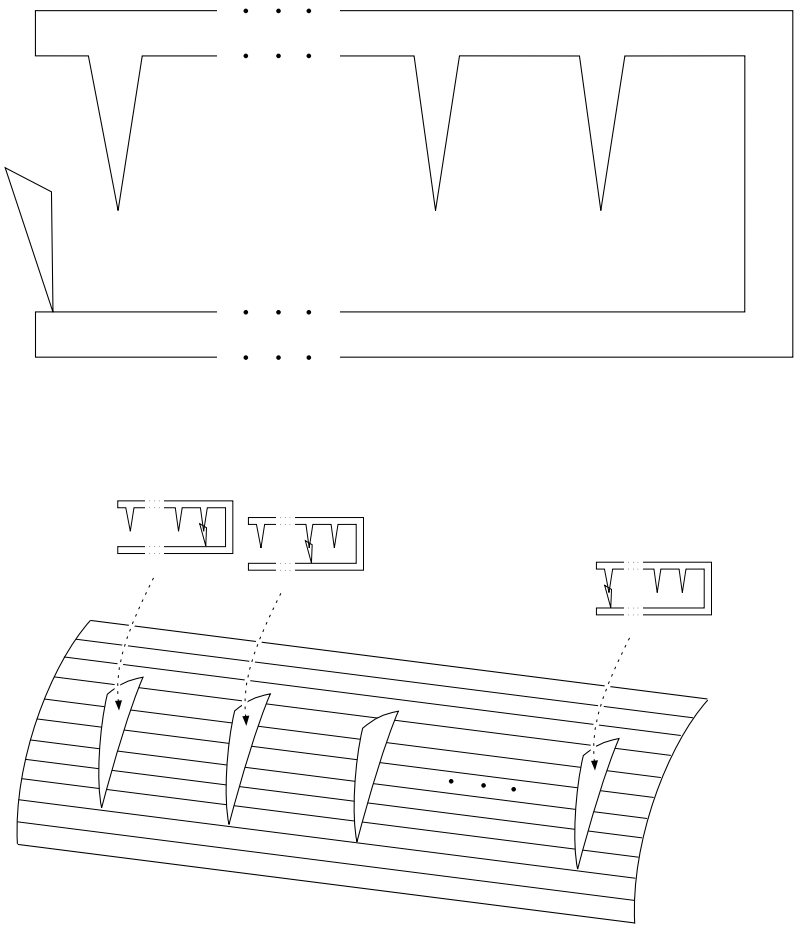


Figure 59: A facet may contain arbitrarily many holes.

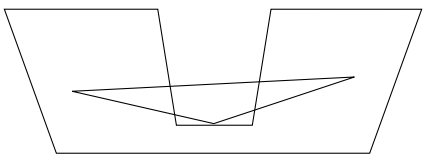


Figure 60: A facet may be completely unreachable.

Representing Configuration-Space Obstacles

In this section we will develop methods for representing configuration obstacles. We will begin by developing a metric representation of obstacle features, and conclude by developing a data structure that encodes the obstacle's metric and topological properties.

Metric Descriptions of Obstacle Features

Each feature of a configuration obstacle may be described by a collection of analytic equations that describe the unbounded set that contains the bounded configuration obstacle feature. For example, the equations for an obstacle facet describe the infinite c-surface containing the facet, and the equations describing an obstacle edge describe the infinite curve containing the edge.

Facets. Configuration-obstacle facets are two-dimensional sets embedded in a three-dimensional space. We can parameterize the facet surface by introducing the perimeter variable p. Figure 61 shows the definition of p for ve and ev facets. The reference point for measuring p is the clockwise vertex of the contact edge; this point is unique for every ve or ev contact. For each facet, the ordered pair (p, θ) provides an unambiguous specification of points on the facet. Our (p, θ) parameterization allows us to write functions that describe the infinite c-surface containing each facet:

$$\begin{aligned} x &= f_x(p, \theta) \\ y &= f_y(p, \theta) \end{aligned}$$

These functions are defined for each facet, and are of the following forms:

$$f_{x_{\mathrm{ve}}}(p, \theta) = k_{1_x} + k_{2_x} p + k_{3_x} \cos(\theta + \kappa_{3_x})$$

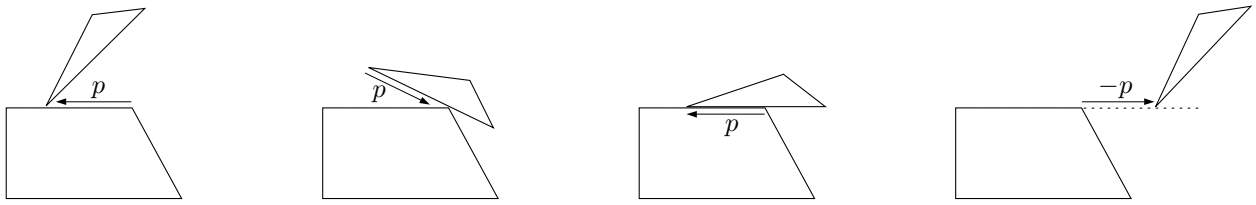
$$f_{y_{\mathrm{ve}}}(p, \theta) = k_{1_y} + k_{2_y} p + k_{3_y} \sin(\theta + \kappa_{3_y})$$

$$f_{x_{\mathrm{ev}}}(p, \theta) = k_{1_x} + k_{2_x} p \cos(\theta + \kappa_{2_x}) + k_{3_x} \cos(\theta + \kappa_{3_x})$$

$$f_{y_{\mathrm{ev}}}(p, \theta) = k_{1_y} + k_{2_y} p \sin(\theta + \kappa_{2_y}) + k_{3_y} \sin(\theta + \kappa_{3_y})$$

where the ve and ev subscripts indicate the functions for ve and ev facets, and the k_i and κ_i terms are constants that depend on the geometry of the facet's contact features (see Appendix 3). We can also define inverse functions that identify the parameter values given a point on the facet c-surface:

$$\begin{aligned} p &= f_p(x, y, \theta) \\ \theta &= f_\theta(x, y, \theta) \end{aligned}$$



$p = 12\text{mm}$ $p = 12\text{mm}$ $p = 11\text{mm}$ $p = -10\text{mm}$

Figure 61: The facet parameter p.

Here f_θ is simply an identity function, and f_p may be computed by a vector dot-product:

$$f_p(x, y, \theta) \quad = \quad \overline{e_{\text{cw}} v} \cdot \hat{e}$$

where e_{cw} is the clockwise vertex of the contact edge, v is the contact vertex, $\overline{e_{\text{cw}} v}$ is a vector pointing from e_{cw} to v, and $\hat{e}$ is a unit vector pointing in the direction of the contact edge. For ve facets, v is a function of configuration; for ev facets, e_{cw} and $\hat{e}$ are functions of configuration. Notice that p is a signed quantity; negative values of p are possible for points on the infinite facet c-surface. However, p is always non-negative for any point on the actual configuration obstacle facet, because of the edge-endpoint limits.

Edges. Because configuration-obstacle edges have only one degree of freedom, they can be parameterized by a single variable. Each configuration-obstacle edge is embedded in an unbounded curve; these curves fall into one of four classes:

$$\begin{aligned}
f_{x_{\text{class-0}}}(p) &= k_{0_x} + k_{1_x} p \\
f_{x_{\text{class-1}}}(\theta) &= k_{0_x} + k_{1_x} \sin(\theta) + k_{2_x} \cos(\theta) \\
f_{x_{\text{class-2}}}(\theta) &= k_{0_x} + k_{1_x} \sin(\theta) + k_{2_x} \cos(\theta) + k_{3_x} \cos(\theta) \sin(\theta) + k_{4_x} \sin^2(\theta) \\
f_{x_{\text{class-3}}}(\theta) &= k_{0_x} + k_{1_x} \sin(\theta) + k_{2_x} \cos(\theta) + \frac{k_{3_x} + k_{4_x} \sin(\theta) + k_{5_x} \cos(\theta)}{k_{6_x} \sin(\theta) + k_{7_x} \cos(\theta)}
\end{aligned}$$

where each k_i is a constant that depends on the geometry of the contact features, and p is similar to the perimeter variable defined for facets. Each x-function has an analogous y-function of the same form. All of these curves are functions of θ except for class-0 curves, which are straight lines that lie in a constant-θ plane. A complete description of class-0 curves requires the θ-value of the containing plane, as well as the $f_x(p)$ and $f_y(p)$ functions that describe the line.

These classes of curves are generic, and some classes apply to more than one type of obstacle edge. Further, a given edge type may give rise to more than one class of curves, depending on the geometry of the contact features. The table below lists the possible curve classes that may arise for each obstacle edge type:

Obstacle Edge Type	Possible Unbounded Curves
ee	class-0
vv	class-1
ve-ve	class-1 two class-0 class-0
ev-ev	class-2 two class-0 class-0
ve-ev	two class-3 class-3 and class-0 class-1 and two class-0

For each edge type, the possible curve classes are listed in order of increasing non-genericity; that is, the most general cases are listed first. Figure 62 shows examples of contact conditions that give rise to each case, along with a depiction of the unbounded curves in the configuration space.

These curves are generated by the intersection of two surfaces in the configuration space. We could use combinations of surface equations to implicitly specify the equations of these curves, but instead we will derive the curve equations directly by treating each edge contact condition as an abstract mechanism, and identifying the mechanism's reachable configurations. Figure 63 shows an example derivation of a curve equation; similar derivations were used to produce the equations given in Appendix 3.

Vertices. We have seen how the metric properties of configuration-obstacle facets and edges may be described by collections of equations; vertices are much simpler. To represent the metric properties of a configuration-obstacle vertex, we simply specify its (x, y, θ) coordinates.

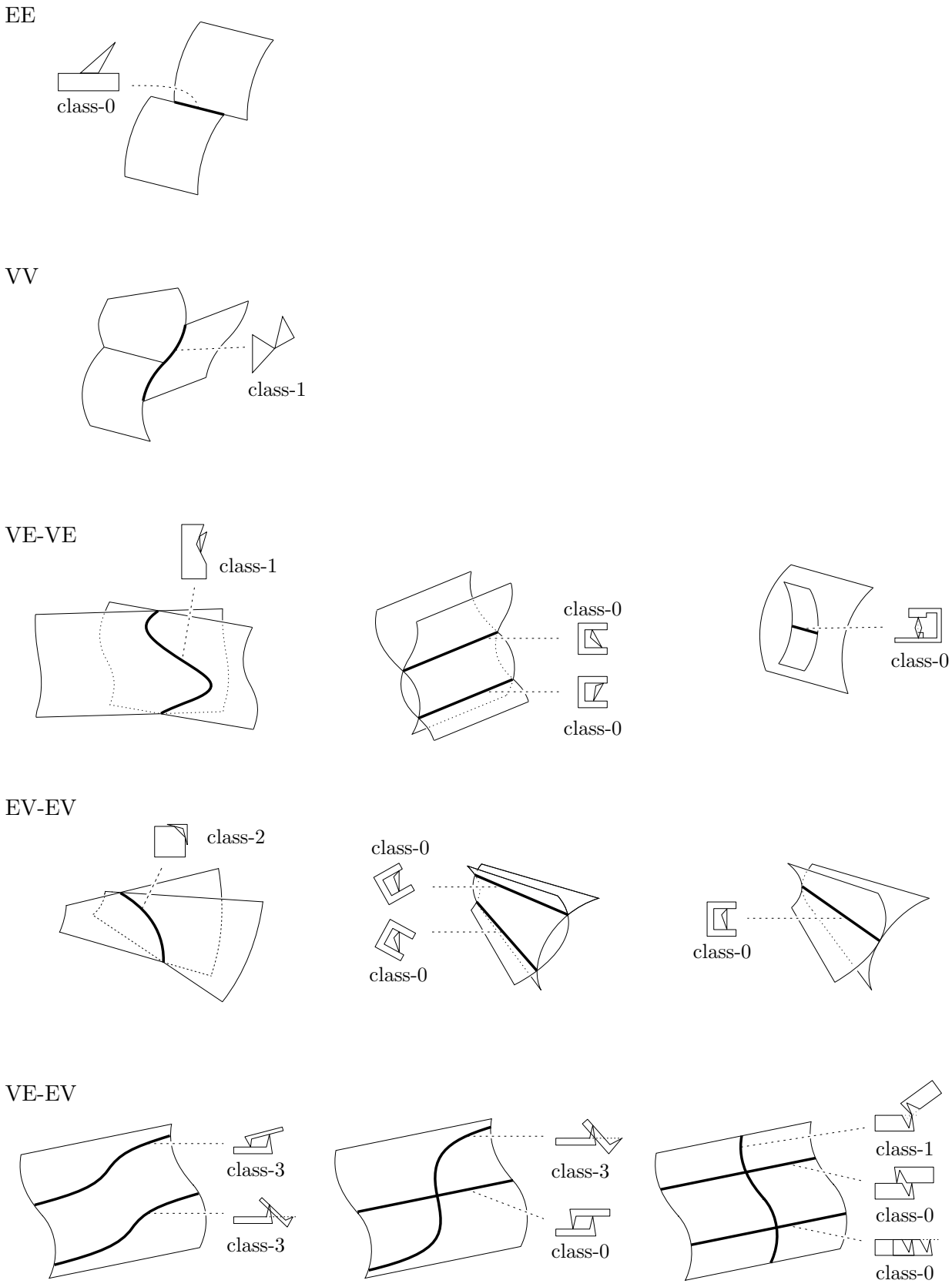
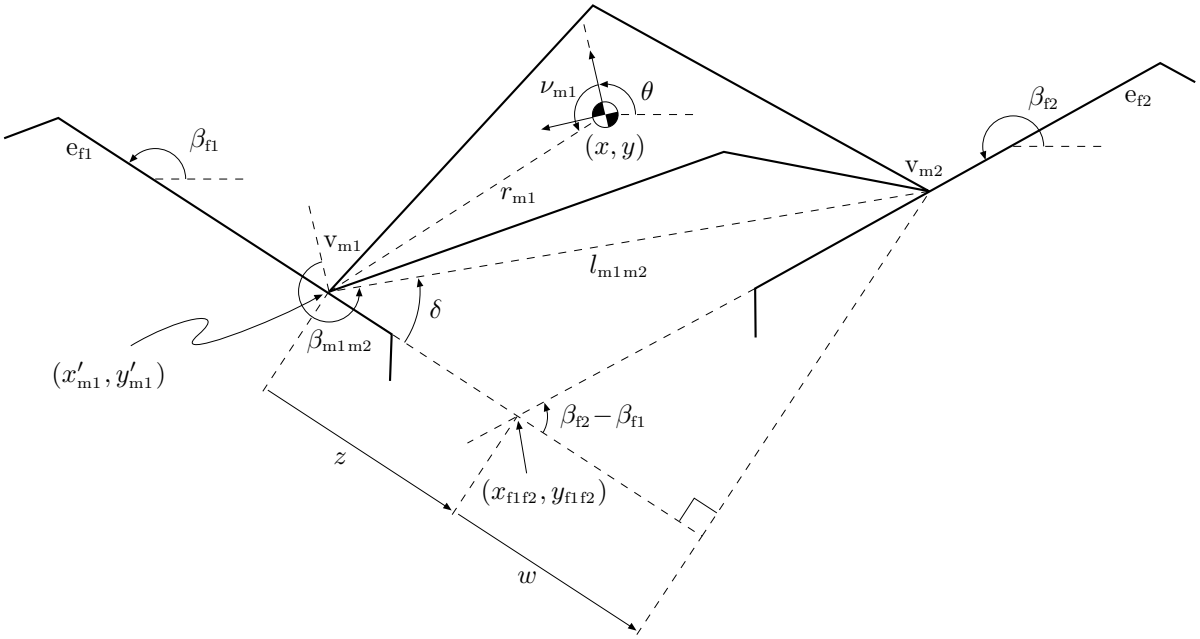


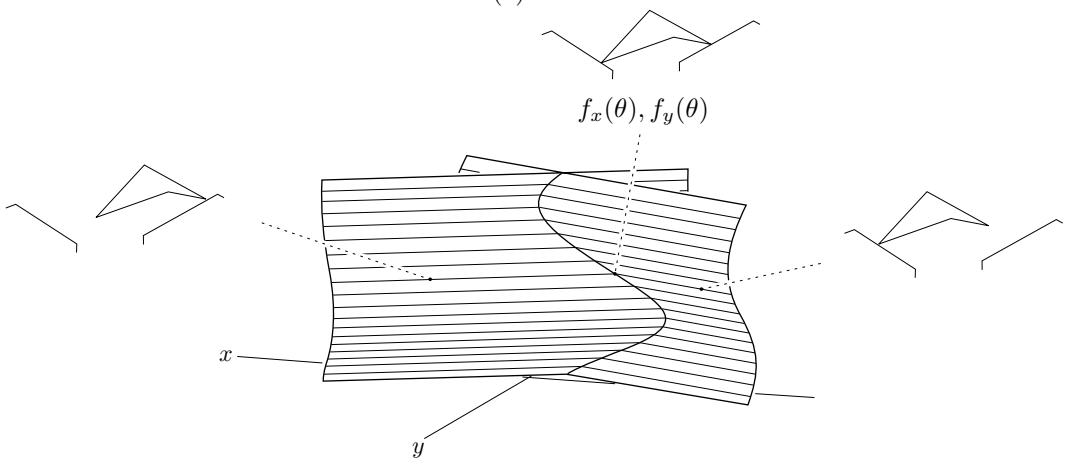
Figure 62: Edge types and associated unbounded curves.



$$\delta = (\theta + \beta_{m1m2}) - (\beta_{f1} + \pi)$$

$$\delta = \theta + \beta_{m1m2} - \beta_{f1} - \pi$$

(a)



(b)

Figure 63: Derivation of the $f_x(\theta)$ function for non-aligned ve-ve contact conditions. (a) The contact condition. Here r_{m1}, ν_{m1}, β_{f1}, β_{f2}, β_{m1m2}, l_{m1m2}, x_{f1f2}, and y_{f1f2} are constants that depend on the geometry of the contact features, and x'_{m1}, y'_{m1}, δ, w, and z are variables that depend on the (x, y, θ) configuration. (b) The relevant c-surfaces and their intersection curve.

$$
\begin{aligned}
x &= x'_{\mathrm{m1}} - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + \mathrm{c}_{\beta_{\mathrm{f1}}} z - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + \mathrm{c}_{\beta_{\mathrm{f1}}} \left[l_{\mathrm{m1m2}}\mathrm{c}_{\delta} - w\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + \mathrm{c}_{\beta_{\mathrm{f1}}} \left[l_{\mathrm{m1m2}}\mathrm{c}_{\delta} - \frac{l_{\mathrm{m1m2}}\mathrm{s}_{\delta}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}} \left[\mathrm{c}_{(\theta+\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\theta+\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}} \left[\mathrm{c}_{\theta}\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \mathrm{s}_{\theta}\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{\theta}\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} + \mathrm{c}_{\theta}\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right] \\
&\quad -r_{\mathrm{m1}}\left[\mathrm{c}_{\theta}\mathrm{c}_{\nu_{\mathrm{m1}}} - \mathrm{s}_{\theta}\mathrm{s}_{\nu_{\mathrm{m1}}}\right] \\
x &= x_{\mathrm{f1f2}} + \mathrm{s}_{\theta}\left[l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\left(-\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right) + r_{\mathrm{m1}}\mathrm{s}_{\nu_{\mathrm{m1}}}\right] \\
&\quad +\mathrm{c}_{\theta}\left[l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\left(\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right) - r_{\mathrm{m1}}\mathrm{c}_{\nu_{\mathrm{m1}}}\right]
\end{aligned}
$$

The transformation between polar and Cartesian coordinates implies that $x_{\mathrm{m1}} = r_{\mathrm{m1}}\mathrm{c}_{\nu_{\mathrm{m1}}}$ and $y_{\mathrm{m1}} = r_{\mathrm{m1}}\mathrm{s}_{\nu_{\mathrm{m1}}}$, so

$$
\begin{aligned}
&f_x(\theta) = k_{0_x} + k_{1_x}\mathrm{s}_{\theta} + k_{2_x}\mathrm{c}_{\theta} \\
&\quad \text{where} \\
&\qquad k_{0_x} = x_{\mathrm{f1f2}} \\
&\qquad k_{1_x} = l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\left(-\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right) + y_{\mathrm{m1}} \\
&\qquad k_{2_x} = l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\left(\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})}\right) - x_{\mathrm{m1}}
\end{aligned}
$$

(c)

Figure 63 (continued): (c) Derivation of the $f_x(\theta)$ function. Here c_{α} and s_{α} are abbreviations for $\cos(\alpha)$ and $\sin(\alpha)$. The functions $f_x(\theta)$ and $f_y(\theta)$ describe a family of class-1 curves; all non-aligned ve-ve edges are embedded in a curve in this family.

A Configuration-Obstacle Data Structure

The previous section showed how collections of analytic equations may be used to describe the infinite surfaces and curves that contain configuration-space obstacle features. In this section, we will show how to construct a topological data structure that represents the boundaries of these infinite sets.

We will begin by examining the toplogical relationships between features on the configuration-obstacle surface. Obstacle edges are segments embedded in unbounded curves; these segments are bounded by a vertex at each end of the segment. Similarly, obstacle facets are embedded in unbounded c-surfaces; these facets are bounded by a loop of edges that surround the facet, and sometimes by additional loops of edges that form holes in the facet. In this way, each obstacle edge and facet is bounded by the lower-dimensional features that surround it.

Similarly, low-dimensional features have higher-dimensional neighbors. Vertices are adjacent to three or more obstacle edges, and edges are adjacent to two neighboring facets. In certain non-generic situations, these basic topological relationships may be violated. Vertices may be isolated or adjacent to only one edge, while edges may have zero or four adjacent facets. When these situations arise, the configuration-obstacle is a non-manifold solid in the sense that there are points on the obstacle whose neighborhoods are not homeomorphic to a disc. In the absence of these situations, the configuration-obstacle is a manifold solid.

The *CO* algorithm reported in this thesis only constructs configuration obstacles that are manifold solids. Issues involved in extending the algorithm to also construct non-manifold solids will be discussed at the end of this chapter.

We can exploit the topological relationships outlined above to define a data structure that represents a configuration-obstacle surface. This data structure is an ensemble of data records connected in a way that encodes the adjacency relationships between obstacle features. These data records are of four types:

c-obstacle The top-level data record, representing the entire obstacle surface.

c-facet A record representing a single obstacle facet.

c-edge A record representing a single obstacle edge.

c-vertex A record representing an obstacle vertex.

The toplevel obstacle record contains an array of facets, and pointers to the polygons that gave rise to the obstacle. The remaining records describe obstacle features; these records contain three principle kinds of information: contact information, metric information, and topological information. In each case, the contact information is the set of polygon features that are in contact. The metric information is a collection of functions and other parameters that describe the metric properties of the feature, and the topological information is a set of pointers that identify adjacent obstacle features. Figure 64 shows the definition of these data records. The *CO* algorithm includes several additional fields to improve its efficiency; these are omitted for clarity.

The *CO* algorithm contructs a data record for each feature of the configuration obstacle. When each record is constructed, the contact-set field is set to describe the feature's contact condition,

and the metric fields are set to describe the unbounded set containing the feature. When the algorithm terminates, the topological fields of each record point to the appropriate records representing adjacent obstacle features. This arrangement of pointers encodes the topology of the configuration-space obstacle, and may be used to identify topological relationships that are not explicitly represented. For example, given a c-facet F and a c-edge E on the facet, the opposite c-facet can be identified by examining the $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields of E, and returning the entry that is not F. Further, all of the c-facets neighboring F may be found by applying this process to all of the c-edges in the edges field of F. Similar procedures may be used to answer other topological queries.

The resulting aggregate data structure is very similar to the winged-edge data structure commonly used to represent plane-faced polyhedra [Baumgart 1974]. Both representations utilize a combination of metric and topological attributes to describe object features, and both representations develop a network of pointers that encode the adjacency relationships between features. However, our configuration-obstacle data structure is different in some key respects: First, the metric information describing configuration-obstacle features is more complex than the linear geometric descriptions used in the winged-edge representation. Second, the orientation of edges is arbitrary in the winged-edge representation, but is meaningful for configuration obstacles since the parameters p and θ impose an implicit orientation on obstacle edges. Finally, we do not sort edges into ordered cycles around faces or vertices; instead, the c-facet and c-vertex records contain an unordered set of adjacent edges.

This section has developed a data structure that represents the metric and topological properties of configuration-space obstacles. The next section will present an algorithm that constructs this data structure, given two input polygons.

c-obstacle

Polygons

moving-polygon fixed-polygon	The input polygons.

Facet Array

facet-array	A three-dimensional array, indexed by a moving-polygon index i, a fixed-polygon index j, and a flag *ve-or-ev* $\in \{0, 1\}$. Each $(i, j,$ *ve-or-ev*$)$ array slot corresponds to a unique ve or ev contact condition. Each slot contains a set of facets that correspond to the slot's contact condition.

c-facet

Contact Information

contact-set	The set of feature-pairs in contact for this facet.

Metric Information

$f_x(p, \theta)$ $f_y(p, \theta)$	These functions return the (x, y) coordinates of points on the facet c-surface, given the parameters p and θ.
$f_p(x, y, \theta)$	This function returns the p-value of a point on the facet c-surface, given an (x, y, θ) configuration-space point. If the input point is not on the c-surface, then the function returns the p-value of the closest point on the c-surface in the specified θ-plane.

Topological Information

edges	The set of obstacle edges bounding this facet. When the facet is only partially constructed, this is the set of candidate obstacle edges that have been posted to the facet.

Figure 64: Data records for representing configuration-space obstacles. Some algorithm control and optimization fields have been omitted for clarity.

c-edge

Contact Information

contact-set	The set of feature-pairs in contact for this edge.

Metric Information

class	One of 0, 1, 2, or 3.
$\theta_{\text{class-0}}$	If this is a class-0 edge, then this is the θ-value of the plane containing the edge.
coefficients_x coefficients_y	If this is a non-class-0 edge, then these are the coefficients of the edge function.
$f_x(p\text{-}or\text{-}\theta)$ $f_y(p\text{-}or\text{-}\theta)$	These functions return the (x, y) coordinates of points on the unbounded curve containing the edge. These are functions of p if this is a class-0 edge; otherwise, they are functions of θ.
$f'_x(\theta)$ $f'_y(\theta)$	If this is a non-class-0 edge, then these functions return the $\frac{dx}{d\theta}$ and $\frac{dy}{d\theta}$ derivatives of the unbounded curve containing the edge, given an input θ-value.
$f_p(x, y)$	If this is a class-0 edge, then this function returns the p-value of a point on the line containing the edge, given an (x, y) point. If the input point is not on the line, then the function returns the p-value of the closest point on the line.

Topological Information

$\text{vertex}_{\text{min}}$ $\text{vertex}_{\text{max}}$	The edge endpoints. $\text{vertex}_{\text{min}}$ is at the end of the edge that has the minimum edge parameter value (p or θ), while $\text{vertex}_{\text{max}}$ is at the end with the maximum parameter value. Zero-length edges are never constructed.
$\text{facet}_{\text{left}}$ $\text{facet}_{\text{right}}$	The facets adjacent to this edge. The left and right directions are defined as follows: If you stand on the surface of the configuration-obstacle at $\text{vertex}_{\text{min}}$ and look toward $\text{vertex}_{\text{max}}$, then $\text{facet}_{\text{left}}$ is on your left.

c-vertex

Contact Information

contact-set	The set of feature-pairs in contact for this vertex.

Metric Information

x	The (x, y, θ) coordinates of this vertex.
y	
θ	

Topological Information

edges	The set of edges adjacent to this vertex.

Figure 64 (continued): Data records for representing configuration-space obstacles. Some algorithm control and optimization fields have been omitted for clarity.

Constructing Configuration-Space Obstacles

This section will present the *CO* algorithm, which accepts a moving-polygon and fixed-polygon as input, and produces a data structure representing the configuration obstacle surface. The *CO* algorithm is structured so that it can construct the configuration obstacle incrementally; in other words, the algorithm can completely construct an arbitrary obstacle facet, and then construct additional facets as desired until the entire obstacle is completed. This feature speeds planning and analysis of manipulation tasks by allowing calling programs to construct only the required portions of the obstacle surface.

The *CO* algorithm is fairly complicated; this is largely due to subtleties in the incremental construction procedures and the presence of optimization strategies that improve the algorithm's average running time. To clarify this explanation, we will first examine a simplified version of the algorithm that does not contain these nuances. The simplified algorithm demonstrates the basic procedures used to identify and construct features of the configuration-space obstacle, and gives an intuitive understanding of the overall computational process. After presenting this simplified algorithm, we will discuss the extensions incorporated in the full *CO* algorithm.

A Conceptually Simple Algorithm

Our simplified algorithm for constructing configuration-space obstacles is summarized in Figure 65. This algorithm constructs every feature that can possibly arise on the configuration obstacle surface, and then deletes features that are buried within the obstacle interior. Except where explictly specified otherwise, the necessary contact, metric, and topological fields are set when each feature is created. When the algorithm terminates, the configuration-obstacle data structure is fully constructed, and contains only features that are actually on the obstacle surface. Each algorithm step will be elaborated below.

Initialize the obstacle. The algorithm begins by creating a c-obstacle data record and initializing the input-polygon fields. The algorithm then uses the number of moving-polygon vertices m and the number of fixed-polygon vertices n to create an $m \times n \times 2$ array with a slot for every ve and ev contact condition. This array is placed in the c-obstacle's facet-array field.

Initialize the facet-array. The algorithm visits each slot of the facet-array, and uses a local geometric test to determine whether contact configurations are possible for the corresponding facet. If the facet's contact vertex is strictly convex, then contact configurations are locally possible, and the facet-array slot is initialized to a set containing a single c-facet data record. The contact-set of the c-facet is set appropriately, and the metric description of the facet's c-surface is constructed. This is accomplished by using the geometry of the contact features to construct the $f_x(p, \theta)$, $f_y(p, \theta)$, and $f_p(x, y, \theta)$ functions that describe the c-surface, using the equations listed in Appendix 3. If the contact is locally impossible, then the facet-array slot is initialized to the null set.

Construct local edges. The algorithm constructs all local edges by forming every possible vv and ee contact between the input polygons, and constructing the local edges consistent with the contact. This process is illustrated in Figure 66. Given a vv or ee contact, the local geometry of the contact features is used to identify the unbounded curve containing the corresponding obstacle edges, as well as the edge endpoints and facet-neighbor relationships. This analysis constructs a collection of c-edge and c-vertex data records that describe the local edges consistent with the contact; these edges are added to their neighboring c-facets.

simplified-CO $(\mathcal{P}_m, \mathcal{P}_f)$

Initialize the obstacle.
Construct the c-obstacle data record, setting the moving-polygon and fixed-polygon fields to $\mathcal{P}_m$ and $\mathcal{P}_f$. Set the facet-array field to an empty array of the appropriate size.

Construct all possible features.

Initialize the facet-array.
Fill each slot of the facet-array with a c-facet data record. Construct the contact and metric information for each facet.

Construct local edges.
Construct the local edges for each facet, and add each edge to its neighboring facets.

Construct conflict edges.
Consider every pair of facets, and call the procedure *simplified-construct-conflict-edges,* which constructs the edges corresponding to the facet intersection.

Construct mc0-edges.
Call the *simplified-construct-mc0-edges* procedure, which identifies sets of superimposed class-0 edges, and merges them to form multiple-class-0 edges.

Connect edge intersection points.
Assure that all intersecting edges are topologically connected; split edges and create or merge vertices as necessary to produce this connectivity.

Remove buried features.
For each facet F:

Remove extraneous edges of F.
Use the local topology of each facet vertex to identify and remove extraneous edges bordering F. When this step is complete, the remaining edges form a collection of closed loops bounding F.

Remove completely buried components of F.
After extraneous edges are removed from F in the previous step, it is still possible that F contains unreachable configurations. This occurs when an entire connected component of F is unreachable. Remove these unreachable components using the following steps:

Assemble edges into loops.
Assemble F's remaining edges into closed loops. Some of these loops will be outermost bounding loops, and others will be nested loops forming holes and islands.

Split F into connected components.
If the loops constructed in the previous step define two or more disconnected components, split F into separate c-facets, each defining a single component that is connected without boundary. Update the facet-array to include the new c-facets.

Discard buried components.
For each connected component of F, choose a point strictly inside the component and construct the corresponding configuration of the input polygons. If the chosen configuration is unreachable, then all configurations within the component are unreachable, so remove the corresponding c-facet from the facet-array.

endfor

Return the obstacle.

Figure 65: The *simplified-CO* algorithm.

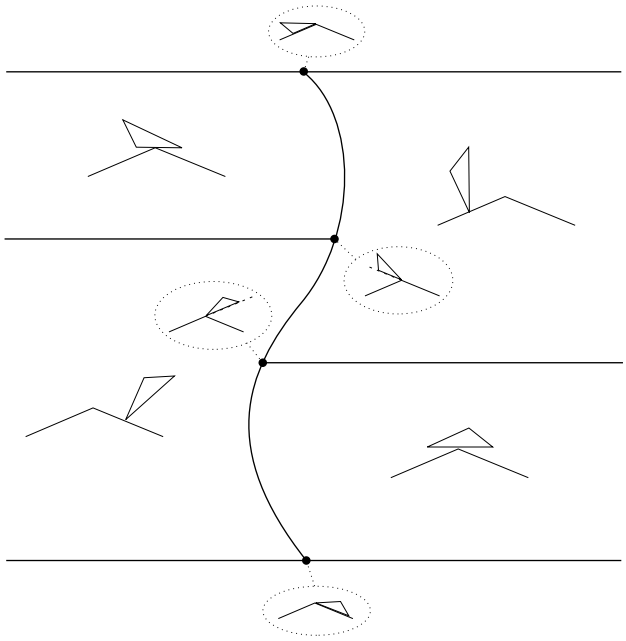


Figure 66: Constructing local edges. A vv contact condition may give rise to several obstacle edges; the endpoints and facet-neighbors of these edges are constructed directly. This figure shows the case of a vv contact with a moving-vertex that is sharper than the fixed-vertex. Four critical configurations are constructed that produce the edge endpoints; these critical configurations correspond to changes in the facets adjacent to the vv edge. Similar diagrams apply when the fixed-vertex is sharper than the moving-vertex, or when either vertex is concave. The construction of ee edges is analogous.

Once this step is complete, the algorithm has constructed all of the facets and local edges that appear on the configuration obstacle. If both input polygons are strictly convex, then the features constructed so far correspond exactly to the features that appear on the obstacle surface. In this case the algorithm may return after connecting the vv and ee edges that share common endpoints. However, if either input polygon is not strictly convex, then additional non-local analysis must be performed; the remainder of this algorithm performs this non-local analysis.

Construct conflict edges. The algorithm constructs the possible conflict edges by considering all pairs of facets, and producing a set of c-edges representing the intersection of each pair. Given a pair of c-facets (F_1, F_2), the c-edges representing the intersection of F_1 and F_2 are constructed using the *simplified-construct-conflict-edges* procedure, shown in Figure 67. When the procedure exits, all possible conflict edges will be constructed and will appear in the edges field of F_1 and F_2. These edges represent the configurations that lie on the intersection of the facet c-surfaces and within the local constraints of both facets.

Construct mc0-edges. When all of the conflict edges have been constructed, some class-0 edges may be superimposed. These coincidences in the configuration-space correspond to non-generic contact conditions. The algorithm identifies these geometric coincidences by applying the *simplified-construct-mc0-edges* procedure, shown in Figure 68. When this procedure exits, all superimposed class-0 edges in the configuration obstacle have been resolved into multiple-class-0 edges, with the $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields set appropriately.

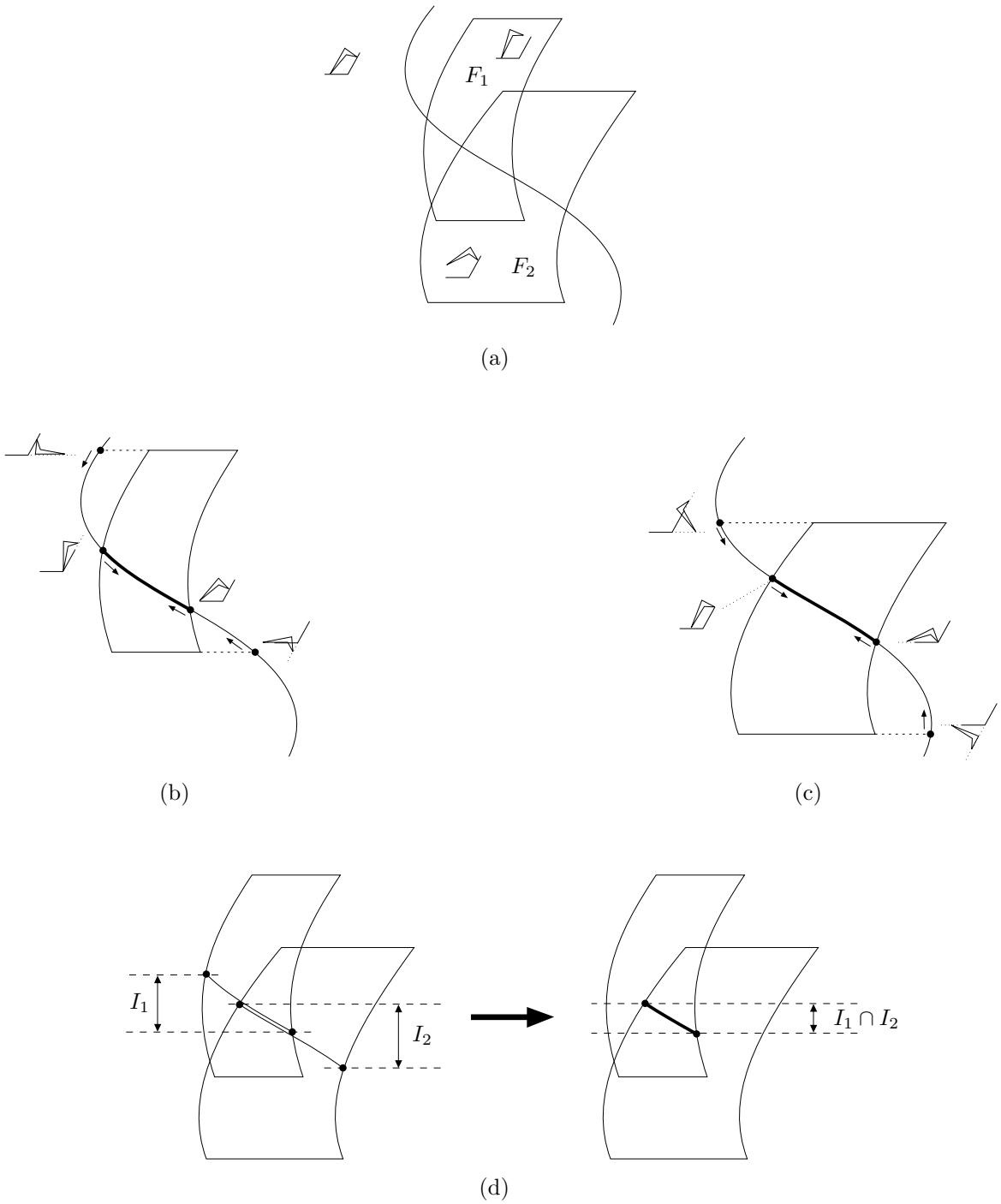


Figure 67: Constructing the conflict edges corresponding to the intersection of two facets F_1 and F_2. (a) Constructing the unbounded curves comprising the intersection of the facet c-surfaces. (b) Identifying the portion of the unbounded curve that intersects F_1. The local constraints of the F_1 contact are used to directly construct a set of minimum and maximum limit points on the unbounded curve; the innermost of these limits delineate the portion of the curve that intersects F_1. (c) Identifying the portion of the curve that intersects F_2. (d) Intersecting the previous results to identify the portion of the unbounded curve that simultaneously intersects both F_1 and F_2.

simplified-construct-conflict-edges (F_1, F_2)

Intersect the c-surfaces.
Construct the set of unbounded curves $\mathcal{C}$ corresponding to the intersection of the F_1 and F_2 c-surfaces, using the kinematic equations listed in Appendix 3. If $\mathcal{C} = \emptyset$, then return.

Analyze the resulting curves.
For each unbounded curve $C \in \mathcal{C}$:

Intersect the curve with F_1.
Use the local geometry of the F_1 contact features to identify a set of intervals $\mathcal{I}_1$ that delimit the curve's intersection with F_1. $\mathcal{I}_1$ is a discrete collection of intervals $I_{1_i} = [p_{\min 1_i}, p_{\max 1_i}]$ or $I_{1_i} = [\theta_{\min 1_i}, \theta_{\max 1_i}]$, depending on whether the curve is class-0 or non-class-0. For every p or θ chosen from an I_{1_i} interval, the corresponding point on the curve lies on or within the local constraints of F_1. If $\mathcal{I}_1 = \emptyset$, then return.

Intersect the curve with F_2.
Similarly, use the local geometry of the F_2 contact features to identify a set of intervals $\mathcal{I}_2$ that delimit the curve's intersection with F_2. If $\mathcal{I}_2 = \emptyset$, then return.

Intersect the previous two results.
Intersect the intervals in $\mathcal{I}_1$ and $\mathcal{I}_2$ to construct $\mathcal{I}$, the set of intervals delimiting curve segments that simultaneously satisfy the local constraints of both F_1 and F_2. If $\mathcal{I} = \emptyset$, then return.

Construct c-edges.
For every interval $I \in \mathcal{I}$:

Create a c-edge data record E.

Set the contact set.
Set E's contact-set to the union of the F_1 and F_2 contact-sets.

Set the metric information.
Set E's metric information fields to the functions describing the unbounded curve C.

Set the endpoints.
Use the minimum value in I and the curve functions to compute the point $(x_{\min}, y_{\min}, \theta_{\min})$ on C; construct a c-vertex at this point, and set E's $\text{vertex}_{\min}$ field to this c-vertex. Similarly, use the maximum value of I to construct a c-vertex at $(x_{\max}, y_{\max}, \theta_{\max})$, and set E's $\text{vertex}_{\max}$ field. Add E to the edges field of both c-vertex records.

Set the facet neighbors.
Set E's $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields to F_1 and F_2, using hand-coded decision procedures that consider the contact geometry to identify the left/right ordering. Add E to the edges field of F_1 and F_2.

endfor

endfor

Figure 67 (continued): The *simplified-construct-conflict-edges* procedure.

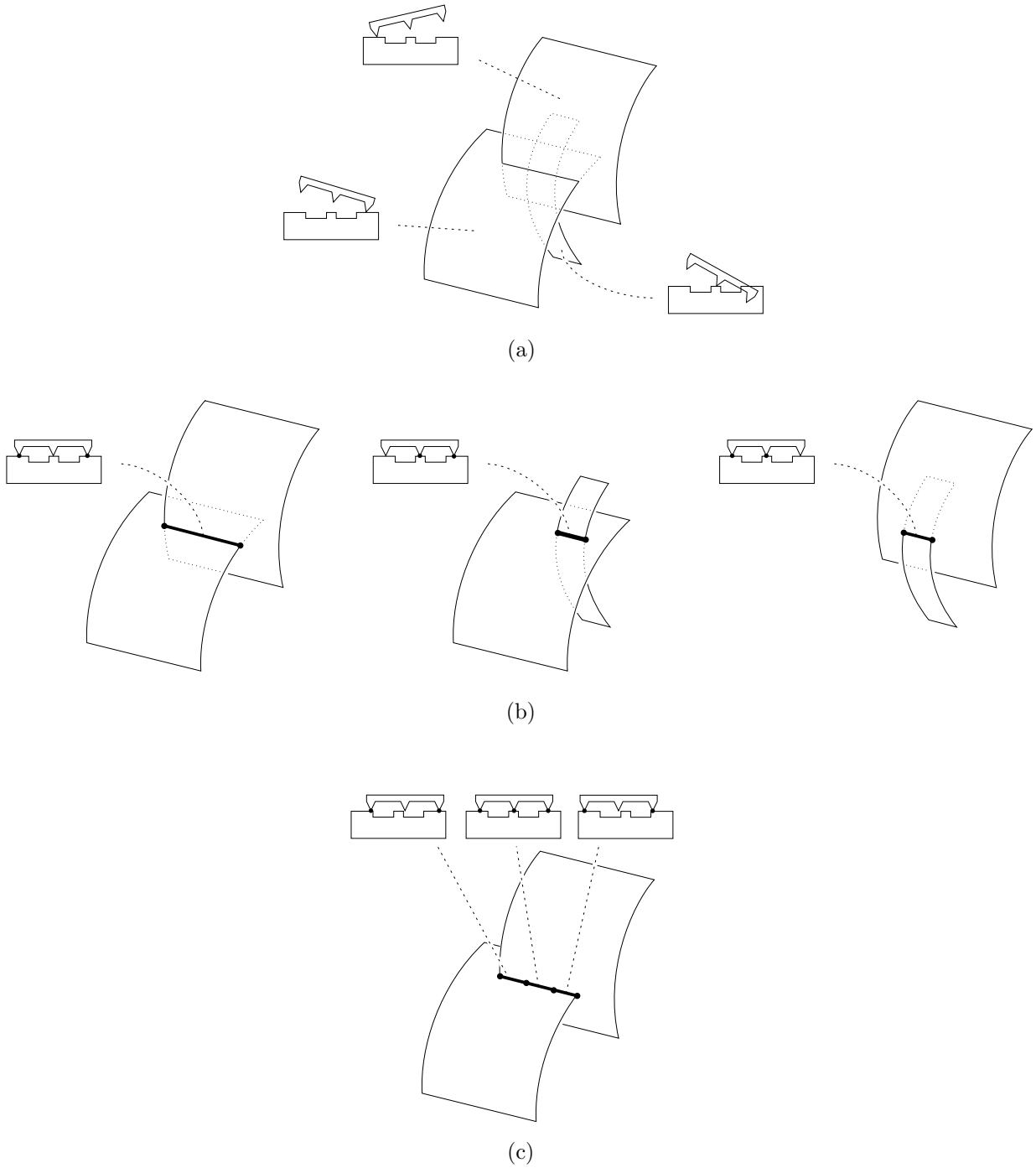


Figure 68: Constructing multiple-class-0 edges. (a) The intersection of three facets which form a multiple-class-0 edge. (b) The three class-0 conflict edges that result from intersecting all 2-combinations of the three facets. The contact-set of each conflict edge is indicated by the highlighted polygon vertices on the label associated with the edge. (c) The result of merging the three conflict edges. A multiple-class-0 edge is formed where the collinear conflict edges overlap; the contact-set of this mc0-edge is the union of the contact-sets of the contributing edges.

simplified-construct-mc0-edges

Form classes of collinear edges.
Assemble all of the class-0 edges that have been constructed, and partition them into classes $\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$ such that all edges within a given class $\mathcal{E}_i$ lie on the same line in configuration space. Superimposed edges can only occur between edges within a given class.

Construct mc0-edges within each class.
For each class $\mathcal{E}_i$:

Topologically merge superimposed edges.
While there is a pair of edges (E_1, E_2) in $\mathcal{E}_i$ that share a finite overlap:

Topologically merge E_1 and E_2, producing a merged edge E_m corresponding to the overlap interval. Split or delete E_1 and E_2 as necessary to properly represent the non-overlapping portions of each edge. If the merge produces two disconnected E_1 edges, add the new edge E_1' to $\mathcal{E}_i$ and the facets neighboring E_1; if the merge eliminates E_1, then remove E_1 from $\mathcal{E}_i$ and the facets neighboring E_1. Apply similar consistency operations to E_2.

Set the contact-set of E_m to the union of the contact-sets for E_1 and E_2.

Set the $\text{facet}_{\text{left}}$ field of E_m to the union of the $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields of E_1 and E_2.

Add E_m to $\mathcal{E}_i$.

endwhile

Convert merged edges into valid mc0-edges.
Now $\mathcal{E}_i$ contains only non-superimposed edges. Some of these edges are the result of merging operations; they are recognizable by their multiple contacts and facet-neighbors.
For every merged edge $E_m \in \mathcal{E}_i$:

Remove E_m from its contributing facets.
The facets that appear in E_m's $\text{facet}_{\text{left}}$ field correspond to the set of facets $\mathcal{F}$ that non-generically intersect to form E_m. Remove E_m from the edges field of every facet in $\mathcal{F}$; this operation and the next step will assure that E_m only appears in the edges field of its neighboring surface facets.

Analyze neighboring facets.
Using the geometry of the non-generic contact condition, identify the facets in $\mathcal{F}$ that lie on the obstacle surface, rather than buried in the interior. There are three possibilities:

Zero surface facets. E_m is a non-manifold spur edge, which we do not represent. Discard E_m.

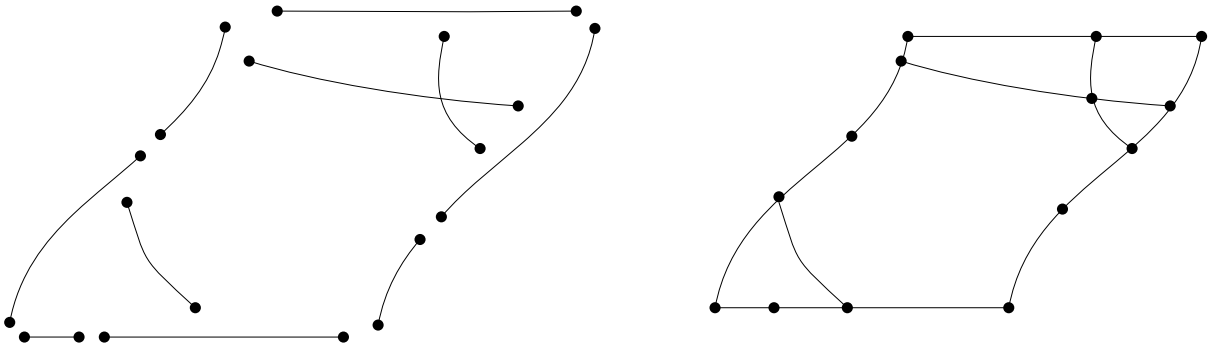
Two surface facets. E_m is a manifold edge, and the left and right adjacent facets F_{left} and F_{right} may be identified. Set E_m's $\text{facet}_{\text{left}}$ field to F_{left}, and set E_m's $\text{facet}_{\text{right}}$ field to F_{right}. Add E_m to the edges field of F_{left} and F_{right}.

Four surface facets. E_m is a non-manifold 4-neighbor edge, which we do not represent. Discard E_m.

endfor

endfor

Figure 68 (continued): The *simplified-construct-mc0-edges* procedure.



(a) (b)

Figure 69: A facet before (a) and after (b) its vertex-connectivity is established.

Connect edge intersection points. Since the previous step resolved all situations where edges overlap for a finite interval, the current configuration obstacle data structure contains edges that can only intersect at a finite number of points. In this step the algorithm assures that for every edge intersection point, there is a c-vertex that topologically connects all of the edges intersecting at the point.

This is accomplished as follows: For every pair of obstacle edges, identify the points where the edges intersect. Then for each intersection point, topologically connect the edges, applying the necessary edge-splitting, vertex-creation, or vertex-merging operations (see Figure 69). Care must be taken to insure that all pairs of edges are intersected, including the new split-edges that are created as a side-effect of the intersection process.

To identify the points where a pair of edges intersect, the algorithm intersects the unbounded curves that contain the edges, and then checks to see if the resulting intersection points lie within the edge endpoints. To intersect the unbounded curves containing the edges, the algorithm applies the following case-based procedure:

- Both edges are class-0: If both edges have the same θ-value, then return the intersection point of the two lines.
- One edge is class-0, and one is non-class-0: Use the θ-value of the class-0 edge to compute a point on the non-class-0 edge. If this point lies on the line containing the class-0 edge, then return the point.
- Both edges are non-class-0: Use the coefficients$_x$ field of each edge to form the equation $f_{x_1}(\theta) = f_{x_2}(\theta)$, which can be written as:

$$0 = k_0 + k_1 \mathrm{s}_\theta + k_2 \mathrm{c}_\theta + k_3 \mathrm{c}_\theta \mathrm{s}_\theta + k_4 \mathrm{s}_\theta^2 + k_5 \mathrm{c}_\theta^2 + k_6 \mathrm{c}_\theta \mathrm{s}_\theta^2 + k_7 \mathrm{c}_\theta^2 \mathrm{s}_\theta + k_8 \mathrm{s}_\theta^3 + k_9 \mathrm{c}_\theta^3$$

  where each k_i is a constant that depends on the coefficients of f_{x_1} and f_{x_2}. Find the roots of this equation as described in Appendix 2; the resulting θ-values correspond to points where the edge $f_x(\theta)$ functions intersect when projected onto the (x, θ) plane. Some of these intersections may be artifacts of the projection; remove these by discarding the θ-values which do not produce equivalent results when substituted into the $f_y(\theta)$ functions of each edge. The role of the f_x and f_y functions may be reversed in this procedure, based on whether the coefficients$_x$ or coefficients$_y$ fields exhibit the maximum spread between the two curves.

Remove buried features. At this point, all of the topological features that appear on the surface of the configuration obstacle have been constructed and connected. The algorithm has constructed every obstacle facet, edge, and vertex that can arise, and the topological connectivity between these features has been identified.

However, features have also been constructed that are buried inside the obstacle; the remainder of this algorithm is concerned with removing these buried features. The algorithm proceeds by visiting each facet in turn, and removing all buried features associated with the facet. When this has been completed for every facet, then the only features that remain are the features that comprise the obstacle surface.

Remove extraneous edges of F. To remove extraneous edges from a facet, the algorithm exploits the property that all facets on the obstacle surface are bounded by a collection of closed edge loops. A consequence of this is that for every vertex V on a loop bounding F, the following is true: If $\mathcal{E}$ is the set of edges adjacent to V, there is a subset $\mathcal{E}_F \subseteq \mathcal{E}$, of edges that neighbor F. Further, $\mathcal{E}_F$ can be partitioned into a set of edge-pairs $\{(E_{1_{\text{cw}}}, E_{1_{\text{ccw}}}), (E_{2_{\text{cw}}}, E_{2_{\text{ccw}}}), \ldots, (E_{n_{\text{cw}}}, E_{n_{\text{ccw}}})\}$, such that for every pair $(E_{i_{\text{cw}}}, E_{i_{\text{ccw}}})$, there are no edges in $\mathcal{E}_F$ that lie between $E_{i_{\text{cw}}}$ and $E_{i_{\text{ccw}}}$, and $E_{i_{\text{cw}}}$ and $E_{i_{\text{ccw}}}$ form clockwise and counter-clockwise boundaries of F with respect to V. This property is illustrated in Figure 70.

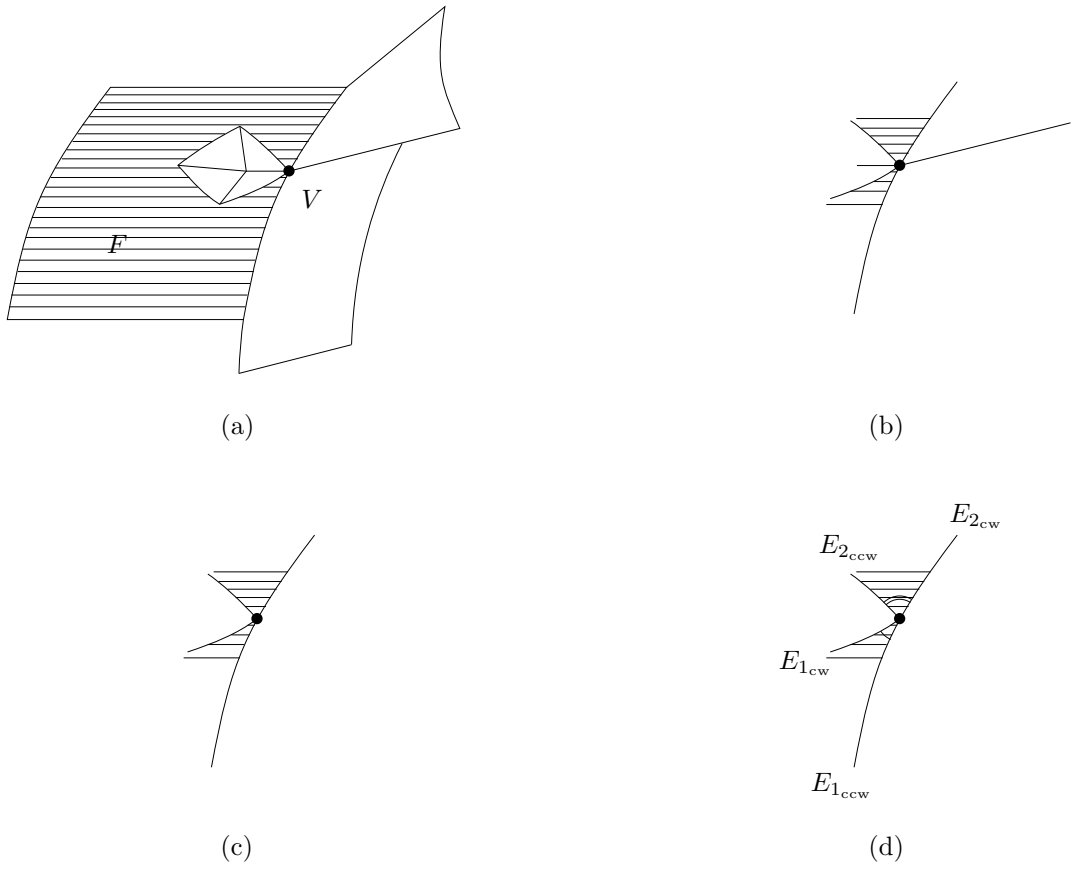


Figure 70: The topology of an obstacle vertex. (a) An example facet F, containing a vertex V. (b) The set of edges $\mathcal{E}$ adjacent to V. (c) The set of edges $\mathcal{E}_F$ adjacent to V that border F. (d) The edge-pairs $\{(E_{1_{\text{cw}}}, E_{1_{\text{ccw}}}), (E_{2_{\text{cw}}}, E_{2_{\text{ccw}}})\}$.

The algorithm uses this property to identify extraneous edges in the following way: For each vertex on F, the algorithm forms the set $\mathcal{E}_F$, and then for each edge $E \in \mathcal{E}_F$, the algorithm constructs a unit vector $\hat{E}_V$ in the facet's (p, θ) space that points away from the vertex tangent to the edge. For class-0 edges, $\hat{E}_V$ is either $[1\ \ 0]^T$ or $[-1\ \ 0]^T$. For non-class-0 edges, $\hat{E}_V$ is computed using a combination of the edge $f'_x(\theta)$ and $f'_y(\theta)$ derivative functions, and the facet parameter function $f_p(x, y, \theta)$.

Once all the $\hat{E}_V$ vectors have been constructed, they are sorted by azimuth and associated with clockwise and counter-clockwise constraints according to their facet neighbor relationship with F. The algorithm then finds the innermost clockwise/counterclockwise pairs of edges; these pairs correspond to edges that are not extraneous at V. After constructing all innermost pairs, the remaining unpaired edges in $\mathcal{E}_F$ are extraneous. Figure 71 illustrates this process. In situations where two $\hat{E}_V$ vectors have equal azimuths, a non-local analysis is necessary to sort the associated edges about V (Figure 72).

After visiting a facet vertex and identifying the extraneous edges, the algorithm then examines each extraneous edge E, and removes the sequence of edges $E\,E'E''E'''\ldots$ that form a singly-connected string of edges bordering F. All of these edges must be removed because the vertices connecting them do not reveal the fact that they are extraneous. Figure 73 illustrates this process.

After applying this procedure to every vertex in F, the edges that remain form a collection of closed loops bounding the facet. Note that in some cases this collection may be null, implying that F contains no reachable configurations.

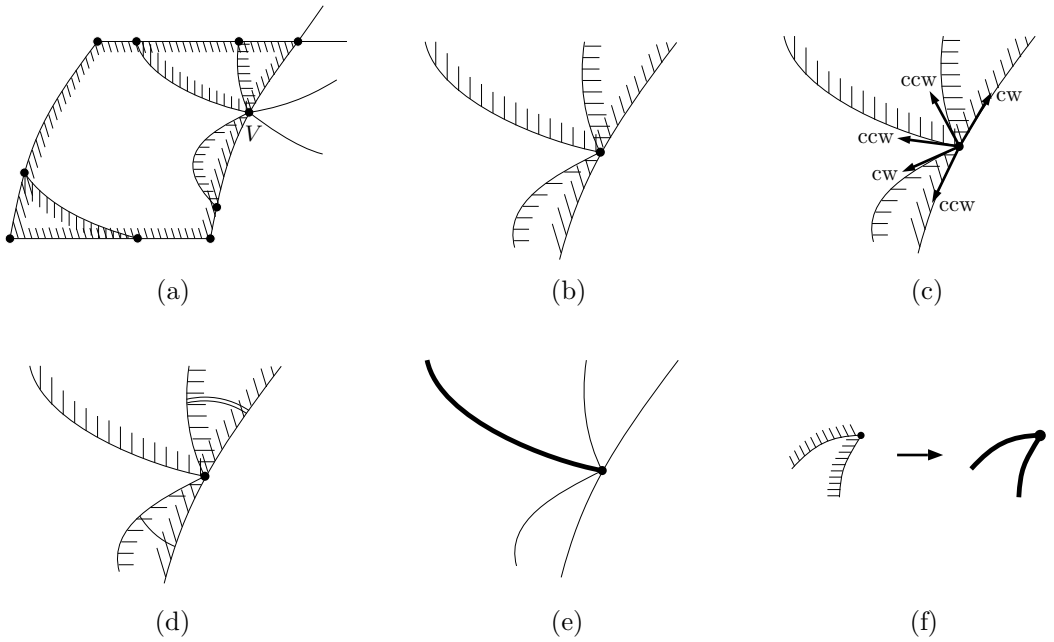


Figure 71: Identifying extraneous edges at a vertex. (a) An example facet F, containing a vertex V. The hash marks associated with an obstacle edge indicate F's left/right adjacency relationship with the edge; edges with no hash marks are not adjacent to F. (b) The set of edges $\mathcal{E}_F$ adjacent to V that neighbor F. (c) After tangent vectors and directional constraints are computed. (d) The edge-pairs found by finding the innermost directional constraints. (e) Edges that are not part of an edge-pair are extraneous; these are shown highlighted. (f) An example where all edges incident to a vertex are extraneous.

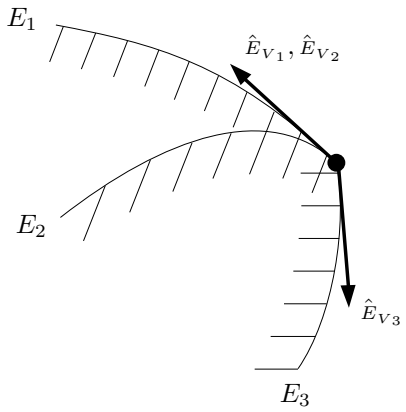


Figure 72: A singular situation arising from two collinear tangent vectors. Sorting $\hat{E}_{V_1}$ and $\hat{E}_{V_2}$ requires additional analysis. The *CO* program includes heuristics for handling these situations, but numerically robust general solutions remain an open research problem.

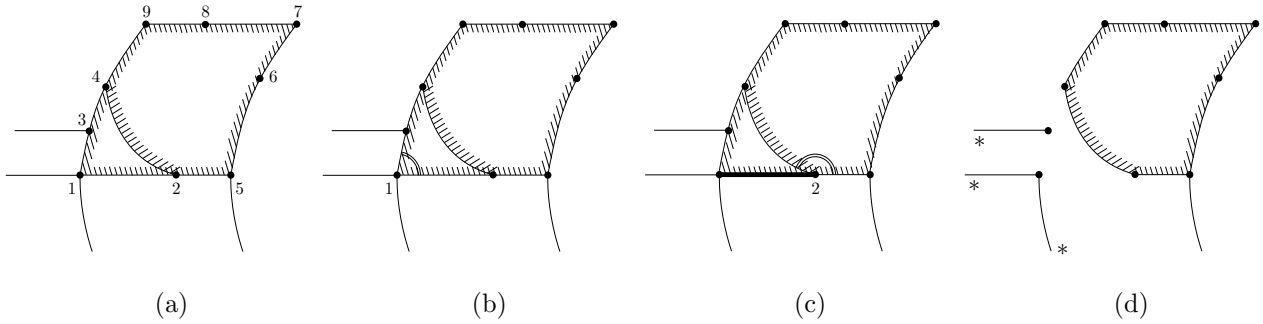


Figure 73: Removing extraneous edges from a facet. (a) A facet F containing local edges and conflict edges, with all edges connected at their intersection points. The hash marks indicate F's left/right neighbor relationship with the associated edge. Some of F's vertices are also connected to edges that do not border F; these edges have no hash marks. The order in which the vertices are analyzed may be chosen arbitrarily; the numerals indicate the order used in this example. (b) Analyzing vertex 1. Two adjacent edges border F, and may be arranged to form an edge-pair. Therefore no edges are locally extraneous at vertex 1, and the algorithm proceeds to vertex 2. (c) Analyzing vertex 2. Three adjacent edges border F; the highlighted edge is extraneous. (d) After removing the extraneous edge and the following sequence of edges that form a singly-connected string of edges bordering F. Notice that the edges that do not border F are ignored. Subsequent processing will ignore vertex 3 because it was deleted from F, and vertices 4-9 will reveal no further extraneous edges. Thus (d) shows the final result of removing extraneous edges from F. The edges marked (*) are also extraneous, and will be removed when their neighboring facets are analyzed.

Remove completely buried components of F. The previous procedures have assured that all of the edges that can possibly arise on the c-surface of F have been created, and that any edges that are not part of a closed loop of edges on F have been removed. Thus F now contains a set of edges that form a collection of closed loops bounding F. These loops may delineate disconnected patches on the surface of F, but since F contains a finite number of edges, the number of separate connected patches is also finite.

In some cases, one or more of F's connected patches is unreachable. Figure 74 shows an example where this occurs. In this example, non-local constraints cause the ve facet to be completely unreachable. However, no conflict edges were created for the facet, because no multiple-contact conditions are possible that are consistent with the facet's local constraints. As a result, the original local constraints remain intact after the extraneous-edge analysis, forming a closed loop of edges bounding the facet.

From this example it is evident that the existence of a closed loop of edges bounding a facet is a necessary but not sufficient condition for concluding that the enclosed facet patch comprises part of the configuration obstacle surface. Therefore, after the algorithm constructs bounded patches of a facet, it must apply additional tests to determine if the configurations contained within each patch are reachable.

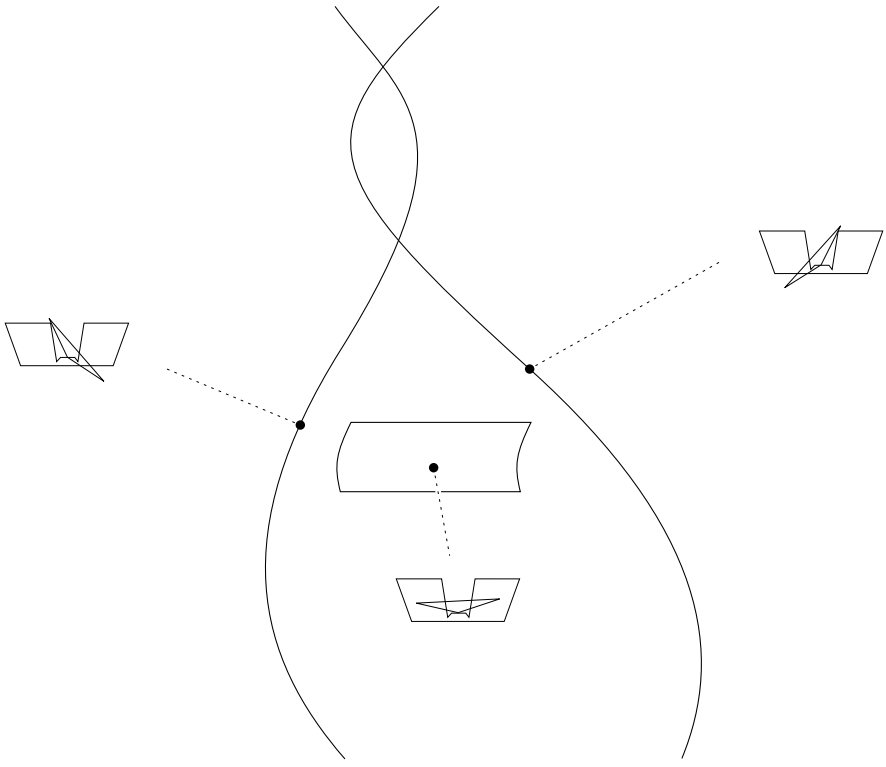
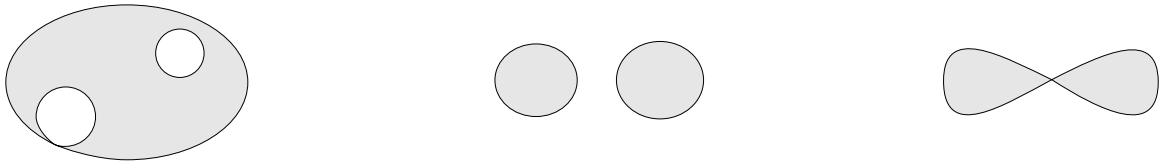


Figure 74: An example of a completely unreachable facet that has no edges removed during its extraneous-edge analysis. The two curves illustrate why no conflict edges were constructed for the facet.



(a) (b) (c)

Figure 75: The definition of a set that is connected without boundary. A continuous set is connected if there exists a continuous path between every pair of points in the set, such that the entire path remains in the set. A set is connected without boundary if the set remains connected when its boundary is not included. (a) A connected set. (b) A set that is not connected. (c) A set that is connected, but not connected without boundary. The set in (a) is connected without boundary.

Fortunately, this may be determined by only checking the reachability of a single point within each connected facet patch. This is because either all configurations within a connected facet patch are reachable, or all are unreachable. This assertion requires that the connected facet patch satisfy the condition of being connected without boundary; this condition is explained in Figure 75.

We can prove this assertion by assuming that the opposite is true, and demonstrating a contradiction. Suppose that the algorithm up to this point has produced a facet patch P that is connected without boundary, and further suppose that P contains a reachable configuration $\mathbf{x}_A$ and an unreachable configuration $\mathbf{x}_B$. Because P is connected, there must be a curve connecting $\mathbf{x}_A$ and $\mathbf{x}_B$ that lies entirely in P. Further, because P is connected without boundary, there must be a curve connecting $\mathbf{x}_A$ and $\mathbf{x}_B$ that lies in P but never touches P's boundary. This curve corresponds to a configuration path in the facet patch that moves the polygons from the reachable configuration $\mathbf{x}_A$ to the unreachable configuration $\mathbf{x}_B$; since this path never touches an edge bounding P, configurations along this path must all be single-contact configurations. But since the path takes a reachable configuration to an unreachable configuration, there must be a transition point where the configuration changes from a valid single-contact configuration to an illegal configuration where the polygons inter-penetrate. Since the entire path is on the facet c-surface and within the local contact constraints, the facet contact features cannot produce this transition. Therefore the transition must occur because some other pair of polygon features cross, producing the illegal configuration. But such a crossing would correspond to a multiple-contact condition, which would give rise to a conflict edge. This implies that all paths in P that connect $\mathbf{x}_A$ and $\mathbf{x}_B$ must touch at least one obstacle edge, which contradicts our assumption that P is connected without boundary. Thus we conclude that it is not possible for the algorithm to produce a connected facet patch that contains both reachable and unreachable configurations.

This allows the algorithm to identify an unreachable facet patch by simply choosing any point on the interior of the patch, and determining whether the corresponding contact configuration is legal. The Boolean outcome of this test will be valid for all points within the patch. The algorithm uses this procedure to identify and discard unreachable connected patches of F; it begins by assembling the edges of F into connected loops, uses these loops to split F into disconnected patches, and then discards each patch that is unreachable. These steps are described below.

(a) (b)

Figure 76: An example looping operation. (a) A complex facet, with all extraneous edges removed. (b) The loops formed for the facet. For clarity, vertices that appear in more than one loop are drawn twice, slightly separated.

Assemble edges into loops. The previous procedures applied to F assure that the edges bounding F form a collection of closed loops. However, these loops are not explicitly represented. The algorithm identifies the loops bounding F by a simple procedure that traces sequences of edges enclosing connected patches of F; this procedure has the desirable feature that it separates patches that are connected, but not connected without boundary.

The loop-finding procedure begins by choosing an arbitrary edge E from the set of facet edges $\mathcal{E}$. The procedure then finds the edge E' that would be encountered next if tracing the boundary of the facet counter-clockwise. Remembering the sequence $E\,E'$, the algorithm then finds the edge E'' that would follow E' if tracing counter-clockwise, and so on until the original edge E is encountered again. The sequence of edges $E\,E'E''\ldots E$ is then assembled into a loop, and these edges are removed from $\mathcal{E}$. If additional edges remain in $\mathcal{E}$, then the algorithm chooses a new seed edge from $\mathcal{E}$ and repeats this process until all edges in $\mathcal{E}$ are exhausted. This procedure places the resulting loops into a set $\mathcal{L}$, and temporarily stores $\mathcal{L}$ in the edges field of F. This procedure is illustrated in Figure 76.

Split F into connected components. The edge-loops described in $\mathcal{L}$ define closed curves that bound F. These closed curves may be recursively nested, producing facet holes, islands within holes, holes within islands within holes, and so on. Here the algorithm uses the metric properties of each loop to identify the nesting relationship between loops. Since the loop-finding procedure assures that two loops L_1 and L_2 do not cross, the nesting relationship between L_1 and L_2 must correspond to one of three cases:

- L_1 contains L_2. Because all obstacle edges are either functions of θ or constant-θ lines, L_1 must contain a vertex not on L_2, and L_2 must contain a vertex not on L_1.
- L_2 contains L_1. As with the previous case, L_1 and L_2 must each contain a vertex that is not on the other loop.
- L_1 and L_2 circumscribe disjoint point-sets. In this case all of the vertices in L_1 may be contained in L_2, and all of the vertices in L_2 may be contained in L_1.

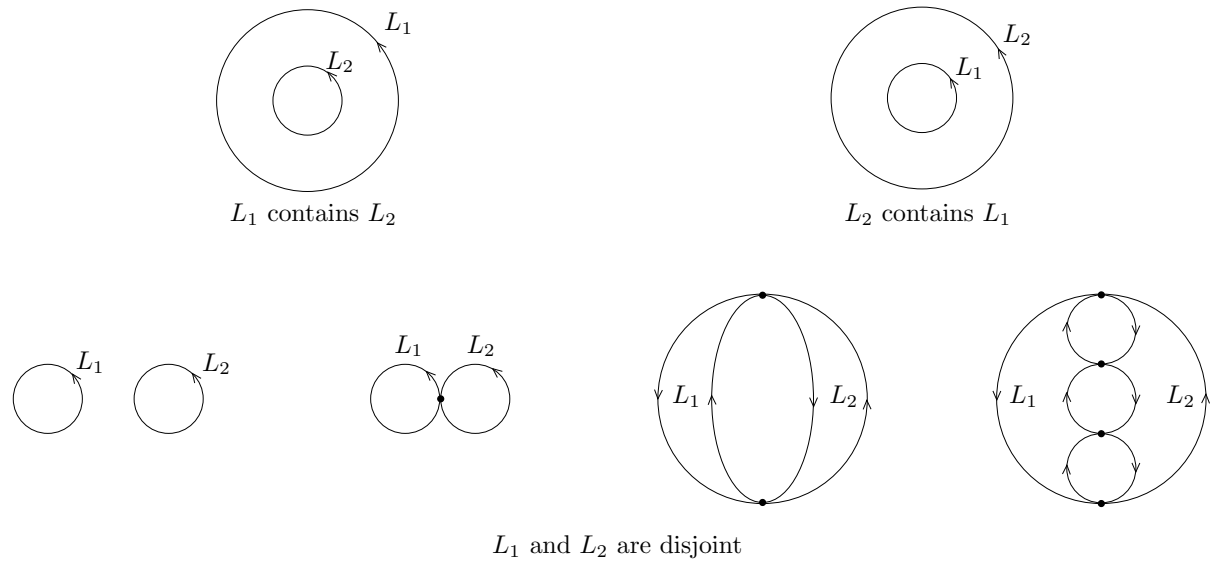


Figure 77: Possible relationships between loops.

These cases are illustrated in Figure 77. The algorithm can distinguish between these cases using the following decision procedure:

> Find a vertex $V_1 \in L_1$ such that V_1 does not lie on any edge in L_2. If no such V_1 exists, then L_1 and L_2 are disjoint.
>
> Find a vertex $V_2 \in L_2$ such that V_2 does not lie on any edge in L_1. If no such V_2 exists, then L_1 and L_2 are disjoint.
>
> If L_1 contains V_2, then L_1 contains L_2.
>
> Else if L_2 contains V_1, then L_2 contains L_1.
>
> Else L_1 and L_2 are disjoint.

Figure 78 shows how the procedure determines whether a vertex V is contained within a loop L.

Using the above decision procedure, the algorithm identifies the nesting relationship between every pair of loops in $\mathcal{L}$, and assembles a collection of trees representing these relationships. Figure 79 shows an example set of loops and the corresponding nest trees.

The algorithm then uses the nest trees to identify the connected patches of F. These patches are immediately evident from the nest trees: The root of each tree corresponds to an outermost bounding loop, and its immediate children correspond to disjoint holes inside this loop. These loops delineate a connected facet patch; further children similarly define other connected patches. The algorithm exploits this outer/inner alternation to identify all of the connected facet patches for each tree; each of these patches will be comprised of an outermost loop and a set of contained holes. Figure 79(c) shows the patches constructed for the example loops.

After identifying all of the connected patches, the algorithm creates a c-facet data record for each patch. To construct a c-facet F' for a given patch P, the algorithm copies F's contact-set and metric description fields into F', and sets the edges field of F' to a set containing the union of the edges that appear in the loops defining P. Performing this operation for every patch produces a set of c-facet data records $\mathcal{F}$ that partition F into connected components; this set is placed in the facet-array slot for F.

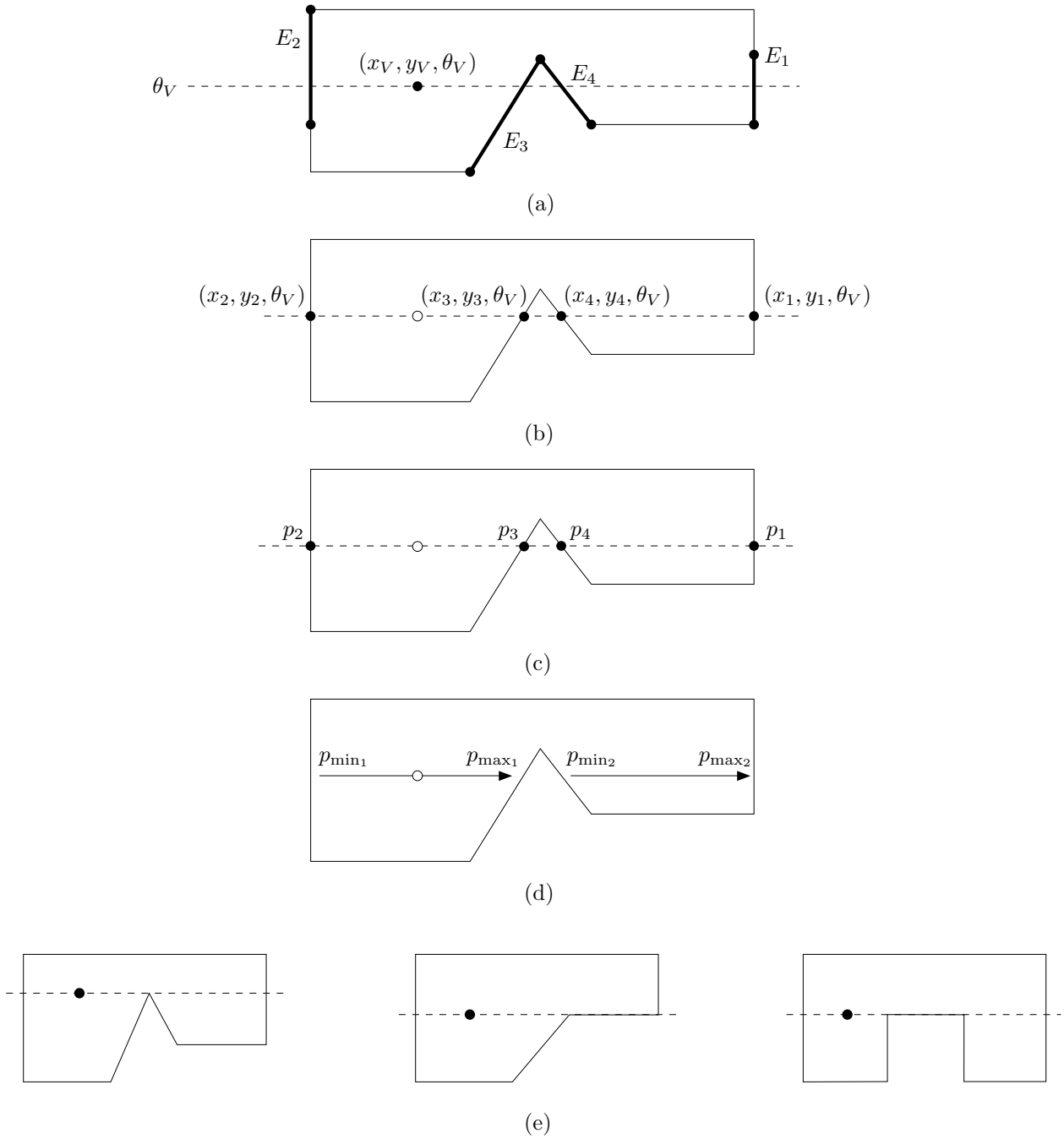
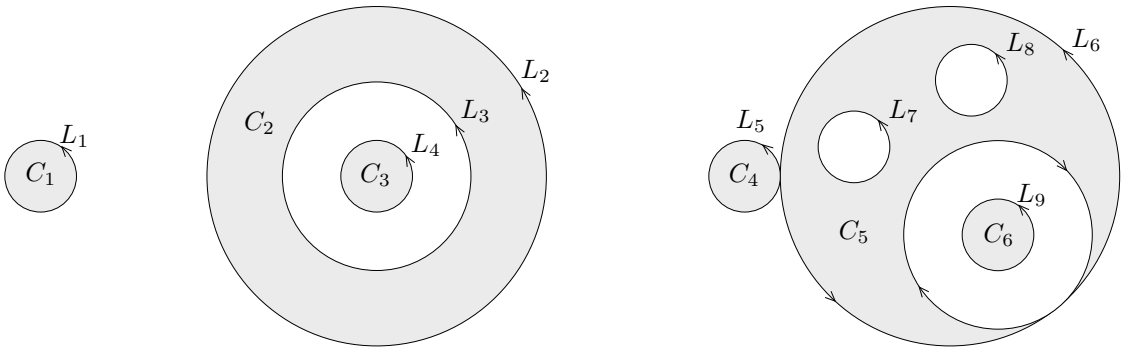
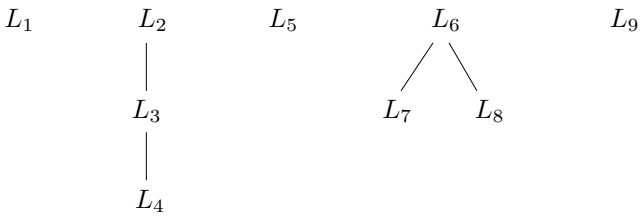


Figure 78: Deciding whether a vertex V is contained in a loop L. (a) Given a point (x_V, y_V, θ_V) on the facet c-surface, the algorithm identifies the set of edges $\{E_1, E_2, E_3, \ldots\}$ that contain θ_V. (b) The algorithm then computes the points $\{(x_1, y_1, \theta_V), (x_2, y_2, \theta_V), \ldots\}$ where these edges intersect θ_V. (c) Then, using F's inverse parameter function $f_p(x, y, \theta)$, the algorithm computes the p-values of these points. (d) These values are then sorted and collected into intervals; if the p-value of the input point is contained in one of these intervals, then V is contained in L. (e) Some example non-generic situations; care must be taken to handle these cases properly.



(a)



(b)

C_1: L_1

C_2: L_2, L_3

C_3: L_4

C_4: L_5

C_5: L_6, L_7, L_8

C_6: L_9

(c)

Figure 79: (a) A complex collection of loops. (b) The collection of trees representing the nesting relationships of the loops. Note that loop L_9 is not contained in loop L_6. (c) The resulting patches that are connected without boundary.

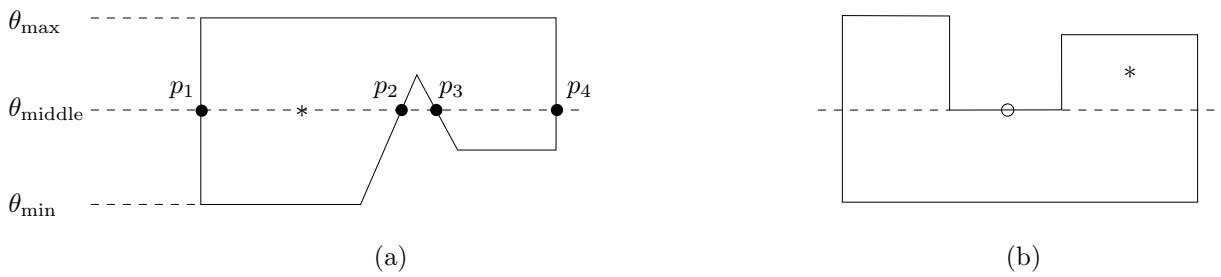


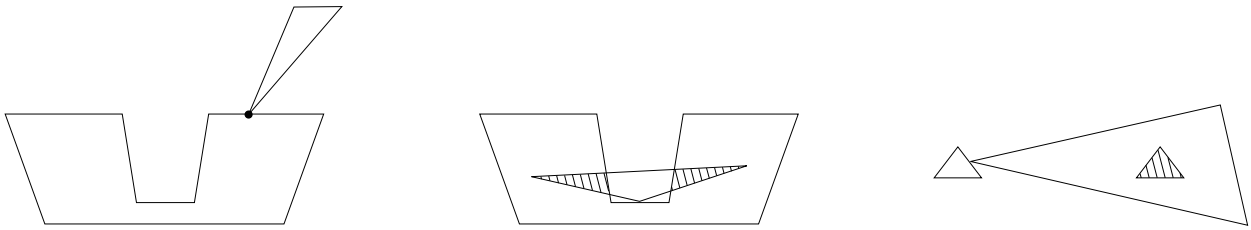
Figure 80: Generating a point in a connected facet patch. (a) A generic situation; the returned point is marked *. (b) A non-generic situation.

Discard buried components. The facet-array slot for F now contains a set $\mathcal{F}$ of c-facet data records that represent patches of F, where each patch delineates a subset of the facet c-surface that is connected without boundary. From the previous discussion, it follows that all configurations within a given patch are either entirely reachable, or entirely unreachable. The algorithm now filters $\mathcal{F}$ to include only patches that contain reachable configurations. This is accomplished by choosing a point strictly inside each patch $F' \in \mathcal{F}$, and determining whether the corresponding configuration is reachable.

Figure 80 shows how the algorithm selects a point from a patch. Given a facet patch F' bounded by a set of edges $\mathcal{E}$, the algorithm checks the endpoints of the edges in $\mathcal{E}$ to find the minimum and maximum θ-values, and then computes the midpoint θ_{middle} of the resulting $[\theta_{\min}, \theta_{\max}]$ interval. Using the same subroutines illustrated in Figure 78, the algorithm identifies the edges in $\mathcal{E}$ that contain θ_{middle}, constructs points on these edges at θ_{middle}, computes the p-values of these points using the inverse parameter function for F', and sorts these p-values to produce a set of p-intervals that define the points on F'. The algorithm then chooses the largest of these p-intervals and computes its mean p_{middle}. This computation identifies a point $(p_{\text{middle}}, \theta_{\text{middle}})$ on F'; the forward parameter functions $f_x(p, \theta)$ and $f_y(p, \theta)$ are then applied to compute the configuration-space point $(x_{\text{middle}}, y_{\text{middle}}, \theta_{\text{middle}})$.

In certain situations the point $(x_{\text{middle}}, y_{\text{middle}}, \theta_{\text{middle}})$ lies on an edge of F' (Figure 80(b)). This is not acceptable, since we require a point strictly on the interior of F'. In these situations the algorithm computes a new θ-value halfway between θ_{middle} and $\theta_{\max}$, and then repeats the above process using the new θ-value. This subdivision is repeated until a suitable point is found.

Once a point strictly on the interior of F' has been identified, the algorithm determines whether the point corresponds to a reachable configuration of the input polygons. This may be accomplished by moving the moving-polygon into the chosen configuration and confirming that the only polygon intersection point is the contact point expected for F' (Figure 81). If the configuration is not reachable, F' is removed from the set $\mathcal{F}$ in the facet-array. After this process has been applied to all patches $F' \in calF$, $\mathcal{F}$ will contain only reachable facet patches.



(a) (b) (c)

Figure 81: Determining whether a configuration is reachable. (a) A reachable configuration; the only intersection between the two polygons is at the contact point, shown highlighted. (b) An unreachable configuration; the shaded regions show unexpected intersection points. (c) One among many special cases that must be considered.

Return the obstacle. After completing the above analysis for every ve and ev facet, the c-obstacle facet-array contains a collection of sets $\{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$, where each $\mathcal{F}_i$ is a set of c-facet data records $\{F_a, F_b, F_c, \ldots\}$ that collectively describe all reachable configurations of the corresponding ve or ev contact condition. Since all possible ve or ev contact conditions are represented by a slot in the facet-array, the ensemble of these c-facets completely spans the obstacle surface. Further, since the construction of each c-facet included construction of its bounding edges and vertices, these lower-dimensional sets are also present in the data structure. Finally, the topological connectivity between features is represented, since this information was constructed and maintained during the algorithm's execution. Thus, the configuration-obstacle data structure is completely assembled at this point, and the algorithm returns the c-obstacle data record.

Correctness of the *Simplified-CO* Algorithm

We can prove the correctness of the *simplified-CO* algorithm by induction. First we will make a preliminary observation regarding the steps that remove buried features from a facet F:

> *Lemma:* If F contains all of its reachable edges at the beginning of the extraneous edge analysis step, then when the remove-buried-components step terminates, the c-facets in $\mathcal{F}$ will *(i)* still contain all of these reachable edges, and *(ii)* all unreachable edges in F will be removed.
>
> *Proof:* Condition *(i)* holds because edges in F are only removed because they are extraneous or part of an unreachable connected patch of F. Since all reachable edges are present in F by assumption, every reachable edge must be part of a closed loop of reachable edges on F. All unreachable edges must lie outside these reachable closed loops, so the extraneous edge analysis will never remove a reachable edge. The remove-buried-components step will also never remove a reachable edge, because every configuration within a closed loop of reachable edges is reachable.
>
> Condition *(ii)* holds because every unreachable edge is either part of a closed loop of unreachable edges, or it is not. If an unreachable edge is not part of a closed loop of unreachable edges, it will be removed during the remove-extraneous-edges step. If it is part of a closed loop of unreachable edges, it will be removed during the remove-buried-components step. ∎

Thus correctness is maintained, as long as all reachable features are present in F_i before the remove-buried-features analysis begins for F_i. Notice that it is not necessary that F_i also contain all unreachable edges; this is important because some unreachable edges may have been removed during the analysis of previous facets. We are now in a position to prove the correctness of the *simplified-CO* algoroithm:

> *Theorem:* When the *simplified-CO* algorithm terminates, the returned data structure contains all of the configuration obstacle features that are reachable, and no features that are unreachable (non-manifold features are excluded).
>
> *Proof:* We proceed by induction, and show that all reachable edges are present in every F_i before the remove-extraneous-edges step is applied to F_i. For the first facet F_0, all reachable edges are present because all possible obstacle edges were constructed during the previous construction steps. Thus the processing of F_0 retains reachable edges and discards all unreachable edges on F_0. For some subsequent facet F_i, if all unreachable edges are present before processing, then by the previous lemma all reachable edges will be retained while all unreachable edges on F_i will be discarded. This completes the induction. ∎

The *CO* Algorithm

The *simplified-CO* algorithm constructs a data structure representing the configuration-space obstacle of two input polygons. The full *CO* algorithm is a modified version of this algorithm, and includes several optimization strategies designed to improve the algorithm's performance. These optimizations are summarized below:

> *Incremental surface construction.*
> Some planning algorithms only require a small portion of the obstacle surface for their analysis. The *CO* algorithm constructs the configuration obstacle incrementally, completing each facet and its surrounding features before proceeding to the next facet. This allows calling programs to specify the facets needed for analysis and planning, thus avoiding the expense of constructing irrelevant features. Additional facets may be constructed later as needed.
>
> *Global convexity checking.*
> In some geometric situations it is possible to determine that a given obstacle facet cannot experience non-local contacts; this occurs when the contact edge and vertex are both *globally convex,* which we will define later. The *CO* algorithm checks this geometric condition and omits the non-local analysis computation when constructing facets that are globally convex.
>
> *Checking for obvious non-intersections.*
> Constructing a configuration-obstacle surface requires the intersection of many pairs of obstacle facets. While constructing the intersection of two obstacle facets is difficult, simple tests can often determine that two facets cannot intersect (for example, if the facets have disjoint θ-intervals). The *CO* algorithm applies a variety of simple tests to decide that two facets do not intersect.
>
> *Reducing the number of edge-intersections.*
> When the local edges bordering a facet are constructed, their connectivity is implicit; connecting these edges at this time reduces the number of edge-intersection decisions required later.

Further, a local edge and conflict edge on a given facet may only intersect at an endpoint of the conflict edge; these edges may be intersected without the full numerical edge-intersection analysis. The *CO* algorithm exploits these and other observations to reduce the number of edge intersections performed.

Parameter caching.
The features of a configuration-obstacle are described by equations whose terms contain parameters describing the input polygons. These parameters often correspond to implicit properties that must be computed from the polygons' (x, y) vertices, such as the length or slope of a polygon edge. The *CO* algorithm caches these parameters to avoid repeated computation of these implicit properties.

These optimizations modify the control structure of the *simplified-CO* algorithm to reduce unnecesary computation. The resulting *CO* algorithm employs the same basic construction procedures as the *simplified-CO* algorithm, but can avoid a tremendous amount of computational effort in some cases. For example, the *CO* program requires 250.6 seconds to construct the entire obstacle for the most complex example in the 1600-case test suite, but is able to construct individual facets of this obstacle in as little as 0.37 seconds, with a median time of 3.24 seconds. Thus if the calling program needs only a few facets to perform its physical analysis, then the computation time may be reduced by up to three orders of magnitude. See Figure 82 for detailed performance data. Because these optimizations involve several intricate details, we will defer further discussion of these optimizations until Appendix 5. This Appendix defines the notion of global convexity, discusses the subtleties of constructing obstacle surfaces incrementally, and presents a pseudo-code summary of the *CO* algorithm.

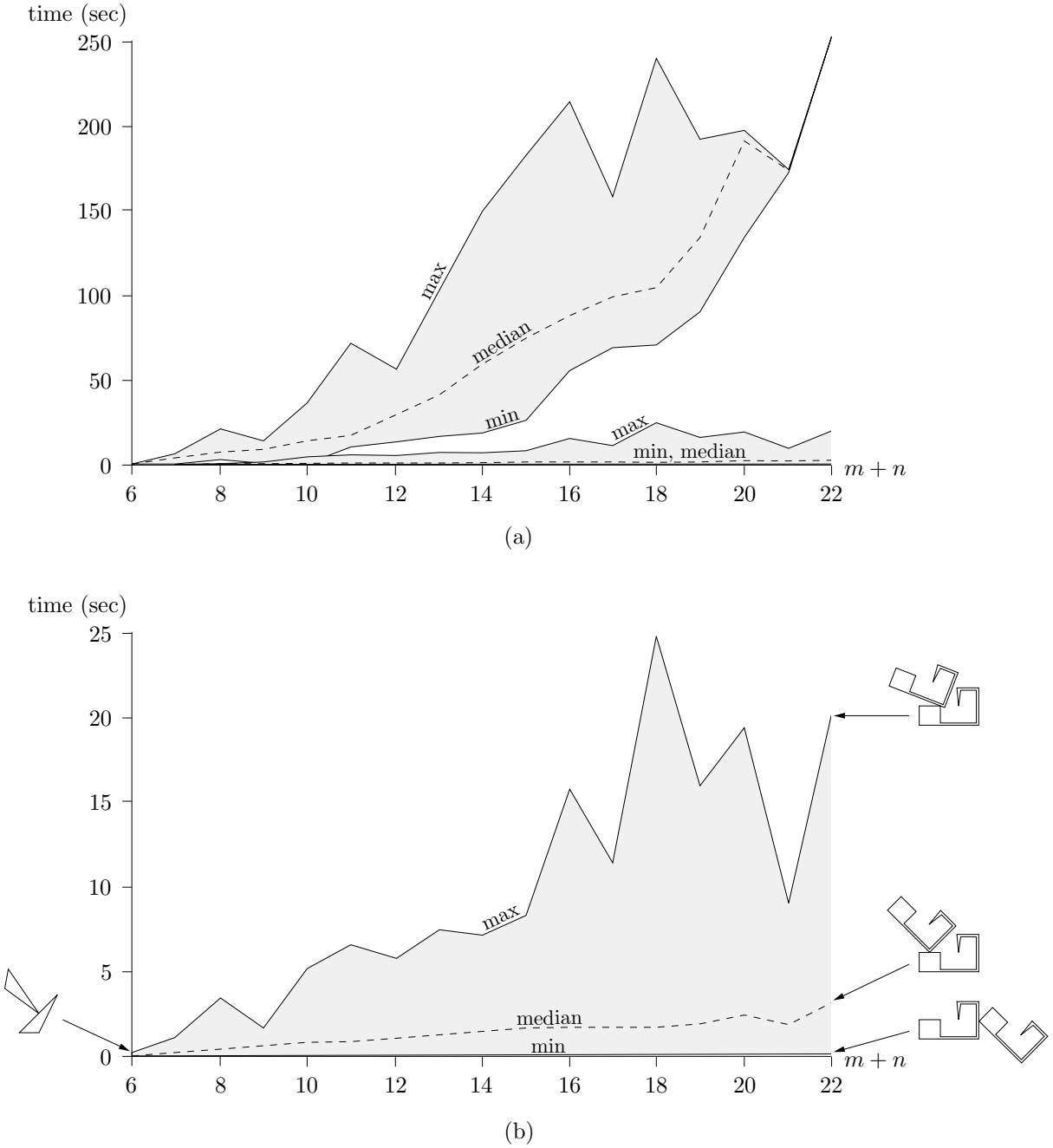


Figure 82: Execution times observed for the 1600-case test suite. The horizontal axis of each plot is the input complexity $m + n$, where m is the number of moving-polygon vertices and n is the number of fixed-polygon vertices. (a) The minimum, median, and maximum execution times required to construct the entire obstacle. Five of the 1600 cases are not shown; these cases required double-precision arithmetic to produce a correct solution, resulting in execution times between 250 and 820 seconds. (b) The minimum, median, and maximum execution times required to construct a single obstacle facet, given the input polygons and an $(i, j, \textit{ve-or-ev})$ facet specification. These times include all required initialization steps, and were measured for every facet of each of the 1600 obstacles. The vertical axis of (b) is magnified compared to (a); the single-facet times are also included in (a) for comparison.

Critique

Comparison with the Avnaim and Boissonnat Algorithm

Avnaim and Boissonnat previously developed an algorithm that is closely related to the *CO* algorithm [Avnaim and Boissonnat 1989]. These algorithms were developed independently, although the Avnaim and Boissonnat algorithm was developed first. This section will compare these algorithms.

Avnaim and Boissonnat's algorithm constructs a representation of the configuration-space obstacle for a pair of interacting polygons, and is designed to support a subsequent search procedure that finds collision-free paths [Avnaim *et al.* 1988a]. The algorithm operates by considering every ve and ev contact C, and applying the following steps:

1. Form a (θ, ξ) parameter space for the contact C. In this parameter space, θ varies over $[0, 2\pi)$, and ξ is a linear parameter that varies over $[0, 1]$.

2. Consider every pair of edges $(\mathrm{e}_m, \mathrm{e}_f)$, where e_m is an edge of the moving-polygon, and e_f is an edge of the fixed-polygon. For each $(\mathrm{e}_m, \mathrm{e}_f)$ pair, use a parallelogram construction to produce a pair of curves in the (θ, ξ) space that delineate a region of configurations that cause e_m and e_f to properly intersect; these curves are functions of θ. The configurations in this region are clearly unreachable because they produce an intersection of the moving-polygon and fixed-polygon edges. The union of these regions for all pairs of edges gives the set of configurations in the (θ, ξ) space that cause at least one pair of polygon edges to properly intersect. This region of illegal configurations is constructed using a plane-sweep algorithm.

3. The complement of the region constructed in the previous step corresponds to the set of configurations where the boundaries of the moving-polygon and fixed-polygon do not cross. The algorithm cuts this complement into subregions ϕ_i, where each ϕ_i is bounded by a single pair of curves limiting its extent in ξ. See Figure 83.

4. The configurations in each subregion ϕ_i may be legal or illegal, depending on whether the polygons are disjoint, or one polygon contains the other. The algorithm chooses a typical point in each ϕ_i region, and determines whether the associated configuration is reachable.

5. The algorithm then considers each reachable subregion and applies a transformation to the subregion to produce a subfacet in the (x, y, θ) configuration space. The resulting ensemble of subfacets corresponds to all reachable configurations for the contact C.

After the algorithm has applied these steps to all of the possible ve and ev contacts, the algorithm identifies the connectivity of the resulting subfacets by considering each type of curve bounding a subfacet, and forming a list of all subfacets containing the type of curve. The resulting subfacets are then sorted, and the subfacet θ-intervals are used to identify adjacency relationships between subfacets. After this process has been applied to all curve types, a complete representation of the configuration-space obstacle has been constructed.

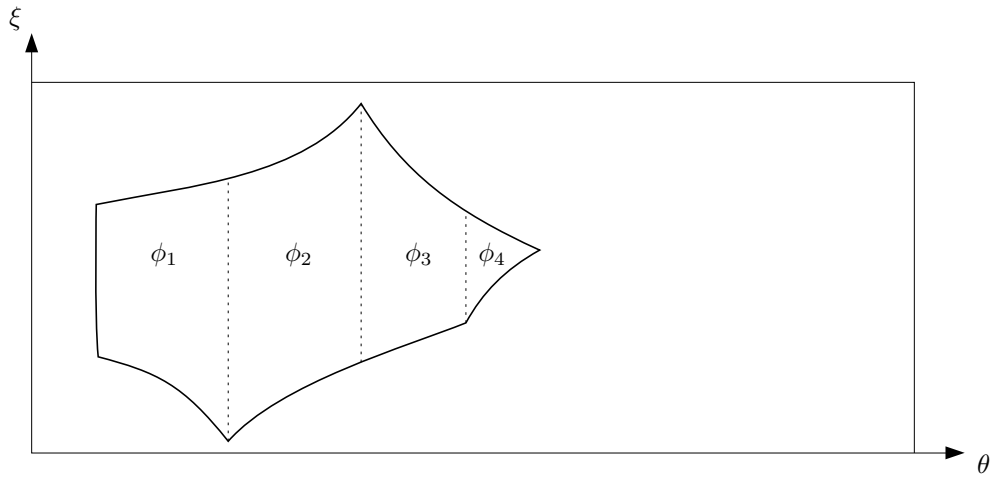


Figure 83: Subregions constructed by the Avnaim and Boissonnat algorithm. The facet subregions ϕ_1, ϕ_2, ϕ_3, and ϕ_4 would be represented by separate data records.

This algorithm is obviously very similar to the *CO* algorithm; what are the differences? Most of the differences are due to the different purposes for which each algorithm was designed; the *CO* algorithm was designed to support physical analysis, while the Avnaim and Boissonnat algorithm was designed to support path-planning. These differences are explored below:

- The first major difference is conceptual; each algorithm employs a different approach to constructing the reachable configurations on a facet. The *CO* algorithm considers intersections of facet c-surfaces, while the Avnaim and Boissonnat algorithm considers intersections of transformed edges in the original space. This is manifested in the facet-construction procedures: The *CO* algorithm identifies unreachable features by constructing edges corresponding to facet intersections, topologically connecting these edges, and then using the vertex topology to discard extraneous edges. In contrast, the Avnaim and Boissonnat algorithm constructs regions that are unreachable because they cause polygon edge intersections; these regions are identified through a plane-sweep algorithm. A preference among these conceptual approaches is left to the taste of the individual reader.

- The *CO* algorithm splits the set of reachable configurations corresponding to a given contact C into subfacets that are connected without boundary. The Avnaim and Boissonnat algorithm produces subfacets that are split further, so that each subfacet is bounded by only two θ-function edges. This causes the Avnaim and Boissonnat algorithm to produce a larger number of facet data records than the *CO* algorithm. In view of the intended application of each algorithm, each of these choices is appropriate: The finer subdivision employed by the Avnaim and Boissonnat algorithm simplifies the search for a collision-free path, while the coarser subdivision employed by the *CO* algorithm is natural for subsequent physical analysis.

- The *CO* algorithm constructs information that is not constructed by the Avnaim and Boissonnat algorithm. This includes the full topological adjacency relationships of generic and non-generic obstacle vertices, and non-generic multiple-class-0 edges. The Avnaim and Boissonnat algorithm does not need to compute this information because it is not required for path planning; however, the *CO* algorithm must compute this information to assure proper physical analysis.

- The *CO* algorithm is explicitly designed to construct the configuration obstacle surface incrementally; the Avnaim and Boissonnat algorithm does not provide this capability. However, extending the Avnaim and Boissonnat algorithm to produce its data structure incrementally would not be very difficult, since each obstacle facet is analyzed individually.

- Both algorithms include unique performance-optimizing features. For example, the *CO* algorithm exploits local consistency constraints to avoid the construction of many conflict edges that are inevitably constructed by the Avnaim and Boissonnat algorithm. On the other hand, the Avnaim and Boissonat algorithm is able to employ a simpler and more efficient test to determine whether each given subfacet is reachable. The asymptotic complexity of both algorithms is the same, and the performance of the implemented programs is virtually identical for constructing full obstacle surfaces. For example, Avnaim and Boissonnat report execution times of 92 and 270 seconds for the two examples presented in [Avnaim *et al.* 1988b]; my *CO* program constructed the configuration obstacle in 116 and 273 seconds for similar input data.

To summarize these differences, each algorithm is strongly influenced by the application for which it was designed. The Avnaim and Boissonnat algorithm constructs a finer division of facets that facilitates subsequent path-planning, and the *CO* algorithm includes incremental construction and analysis of non-generic obstacle topology to support subsequent physical analysis. Each algorithm contains features that could be used to improve the other, and further study of both algorithms could lead to a hybrid algorithm that is superior to each of these results.

Implementation and Robustness Issues

I implemented the *CO* algorithm and performed experiments to measure its robustness and efficiency. The resulting program is comprised of about 9600 lines of Common Lisp code, excluding substantial debugging and rendering routines.

This program is an approximation of the *CO* algorithm in several respects. First, the program performs its numeric calculations using floating-point arithmetic; this finite-precision limitation leads to inaccuracies in the metric representation of obstacle features. Second, certain procedures in the program do not exactly match the procedures described here; for example, the procedure that identifies closed edge-loops uses a transitive-closure method that will fail to separate facet patches that are connected, but not connected without boundary. These latter differences between the algorithm and the program logic are relatively minor, and are chiefly due to insights I gained while writing this document.

However, the use of finite-precision arithmetic produces a much deeper discrepancy between the ideal *CO* algorithm and the implemented program. Because of the finite precision of floating-point arithmetic, small errors pervade the geometric description of obstacle features; these errors can lead to failures in simple numeric decision predicates (such as =), with concomitant errors in the constructed obstacle topology. In an effort to improve the robustness of the *CO* program, I included several approaches to reducing the impact of these errors:

- *Fuzzy arithmetic.* All numeric decision procedures in the *CO* program use fuzzy arithmetic; that is, two numbers are equal if they are within a certain tolerance, *et cetera.* I took great care to apply these tolerances as consistently as possible; this required a fair amount of thought when analyzing nonlinear geometric features.

- *Utilize raw geometry.* In an attempt to minimize the effects of numerical error propogation, the *CO* program attempts to base its decisions on the original polygon geometry as much as possible. For example, endpoints of local and conflict edges are computed using the local contact features, rather than through intersection of configuration-obstacle features. Depending on the specific feature geometry, this can be a mixed blessing (see below).

- *Explicitly include non-generic cases.* The *CO* program includes specific code to detect and handle degenerate geometric cases whenever possible. This led to very tedious programming, but contributed strongly to the program's overall robustness.

- *Compute edge intersections numerically rather than analytically.* Analytical methods may be used to directly indentify the intersection points of obstacle edges in all but the most complex cases; however, all of the analytical methods that I implemented proved to be numerically brittle, and the current program uses a robust numerical procedure for all edge intersections (see Appendix 2).

This combination of approaches proved to be fairly successful. I tested the program using a suite of 1600 test cases, comprised of all pairs of the 40 polygons displayed in Figure 14; 35 of these polygons were hand-designed to assure that all of the possible edge types arose, and the remaining 5 polygons were randomly-generated to reduce the presence of human bias. Appendix 6 gives the coordinates of these polygons. The 1600 test cases were given to the program, and post-processing tests were applied to check the validity of the program output. These tests assured that the returned data structure represented a non-null, closed, orientable manifold surface, and that all nested surface shells were properly oriented. These topological tests were augmented by metric tests that chose a sampling of points on the obstacle surface, and assured that the corresponding configurations were reachable. In addition to these post-processing tests, run-time checks were added to test certain critical invariants; for example, an error would be flagged if an edge was ever deleted from a facet whose remove-buried-features step had already been completed. After some debugging effort, all 1600 cases passed these tests without an error.

These results are encouraging, but weaknesses still remain. For example, the polygon shown in Figure 84 consistently causes the program to fail. These failures occurred because the two edges marked (*) are very nearly parallel, but not aligned closely enough to be considered parallel. Consequently these edges give rise to conflict-edges that are nearly horizontal. These non-class-0 edges are numerically sensitive; small errors in θ give rise to large errors in x and y. Consequently, small errors in the *CO* algorithm's interval analysis lead to large errors in the constructed edge endpoints.

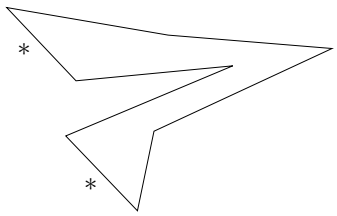


Figure 84: A polygon that causes the *CO* program to fail. The edges marked * have orientations separated by just 0.048°; these nearly-parallel edges give rise to numerically sensitive conflict edges.

This problem is symptomatic of the singular nature of c-surface intersections. An inspection of Figure 62 reveals that singular transitions between class-0 and non-class-0 edges can occur for a variety of gemetric conditions. These singular transitions correspond to dramatic changes in the representation and analysis procedures employed by the *CO* algorithm. Given that class-0 and non-class-0 edges are manipluated in very different ways, it is not surprising that problems occur near the boundary separating these two cases.

One can imagine several approaches to solving this problem. For example, condition numbers could be applied to select the most robust analysis procedures for a given geometric situation. Another approach might be to formulate the metric representation of the obstacle features in a different space with more favorable numerical properties; the formulations employed by [Avnaim and Boissonnat 1989] or [Canny 1986] come to mind as possible alternatives. A third approach might be to use a more robust scheme for representing and manipulating obstacle surface elements; new results in modelling planar polyhedra might be applicable to this domain as well [Milenkovic 1988; Hoffmann 1989; Sugihara and Iri 1989]. Which approach will ultimately yield the most practical and robust programs for constructing configuration-space obstacles remains an open question.

Extending the Algorithm to Include Non-Manifold Obstacle Features

The *CO* algorithm will construct a configuration obstacle for an arbitrary pair of input polygons; however, if the polygons give rise to non-manifold features on the configuration obstacle, these features will be omitted in the output data structure.[4] To extend the *CO* algorithm to include non-manifold obstacle features, several modifications are required. We will address the modifications required for non-manifold edges and vertices separately.

Including non-manifold edges is relatively straightforward. For example, the *construct-mc0-edges* procedure already constructs non-manifold multiple-class-0 edges, but discards them; this procedure could be modified to retain these edges. Retaining these non-manifold edges would require additional modifications to assure correctness; for example, the procedure that identifies the extraneous edges at a vertex would have to be modified to handle edges that contain the current facet in both the $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields of a given edge. Care must be taken to properly handle

[4] An exception is the case shown in Figure 51(e), which gives rise to a non-manifold 4-neighbor edge that is not superimposed with any other class-0 edge. Special code must be included to address this case, or errors may result.

non-manifold edges that are not multiple-class-0 edges, such as the case illustrated in Figure 51(e). Several data structure modifications would also be required; for example the c-edge $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields must be redefined to properly represent the adjacent facets for spur and 4-neighbor edges. Further, the c-obstacle data record must be extended to contain a field for isolated spur edges, since these edges may be completely disconnected from all obstacle facets.

Including non-manifold vertices is more challenging. Some of the modifications are similar to the modifications required for non-manifold edges — for example, extra fields must be added to the c-obstacle and c-facet data records to store isolated vertices. However, recognizing these non-manifold vertices is a more fundamental problem. Since the topological construction procedures and reachable/unreachable decision procedures are predicated on the existence of a finite range of configurations, these non-manifold vertices may fall through the cracks of the analysis. One possible approach would be go ahead and construct zero-length conflict edges, and include them in the analysis in some consistent way. This may lead to an algorithm that can detect non-generic coincidences in obstacle vertices, and efficiently identify non-manifold vertices on the obstacle surface. To recognize isolated obstacle vertices, we could add a step that checks the reachability of every discarded vertex; this test would be expensive, but it might be difficult to generate the corresponding form-closure configurations any other way. In all of these extensions, numerical robustness will be a critical concern.

Statics

Friction and Planar Forces

One of the simplest and most widely-used models of dry friction was proposed by Coulomb in the eighteenth century [Coulomb 1781]. This model asserts that the maximum lateral force F_f that may be generated by a frictional contact is proportional to the normal force F_N; this gives rise to the familiar friction equation $F_f \leq \mu F_N$, where the friction coefficient μ depends on the type and surface condition of the materials interacting at the contact. Empirical studies have shown that this model is realistic for dry contacts moving at low speeds, especially if μ is considered an uncertain parameter. This chapter adopts Coulomb's model, and treats μ as an interval of possible values $[\mu_{\min}, \mu_{\max}]$, where these bounds are obtained by observation of both kinetic and static frictional interactions between the materials of interest.

The $F_f \leq \mu F_N$ specification of Coulomb's law is useful for algebraic analysis, but geometric interpretations of the law are more convenient for some problems. One common interpretation is the *friction cone,* first described by Moseley [1835]. The friction cone expresses Coulomb's law in the (F_x, F_y) force space, as shown in Figure 85. In this figure, a particle is in contact with a frictional constraint, and the friction cone expresses the set of possible contact reaction forces.

An example applied force F_a is shown, both in the original space and in the (F_x, F_y) space. Since F_a points opposite and into the friction cone, it is possible for the contact to generate a reaction force that balances F_a. Thus static equilibrium is possible for the situation shown in Figure 85.

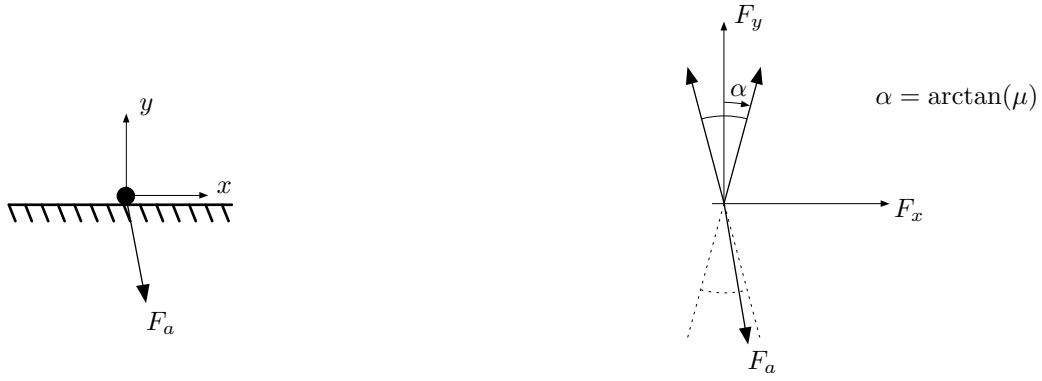


Figure 85: The friction cone [Moseley 1835]. The dashed lines indicate the negated friction cone.

Figure 86: Since the applied force F_a lies outside the negated friction cone, the particle accelerates toward the right.

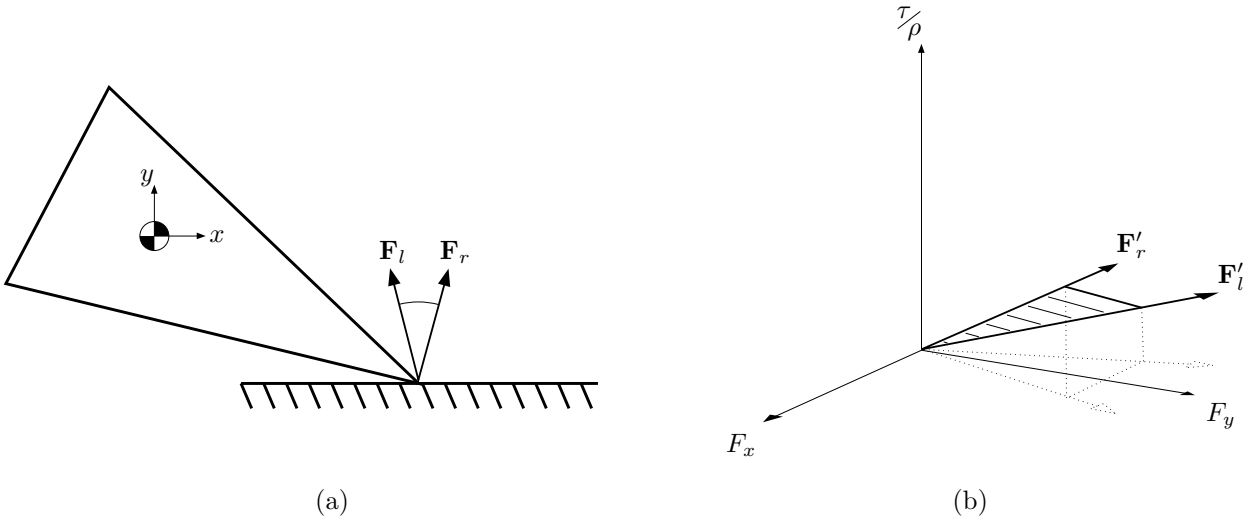


Figure 87: The configuration-space friction cone [Erdmann 1984]. The friction cone rays $\mathbf{F}_l$ and $\mathbf{F}_r$ both exert a positive torque about the polygon center of mass, so the c-space friction cone rays $\mathbf{F}'_l$ and $\mathbf{F}'_r$ are both above the (F_x, F_y) plane in the force-torque space. $\mathbf{F}'_r$ is elevated higher than $\mathbf{F}'_l$ because the ray $\mathbf{F}_r$ exerts a larger torque than $\mathbf{F}_l$.

However, if F_a did not point opposite and into the contact friction cone (as in Figure 86), then no contact reaction force may balance the applied force, and static equilibrium is impossible. These situations are distinguished by the relationship between F_a and the negated contact friction cone; static equilibrium is possible only when the applied force lies within this cone.

So far we have considered the case of a particle experiencing a frictional contact; frictional contacts applied to extended bodies are more complex. Figure 87(a) shows a contact between a polygonal object and a frictional surface. As in the case of a particle, a friction cone may be used to illustrate the set of reaction forces that may be exerted at the contact. However, because the contact point is not collocated with the polygon's center of mass, these contact reaction forces exert torques about the center of mass that were not present in the particle case.

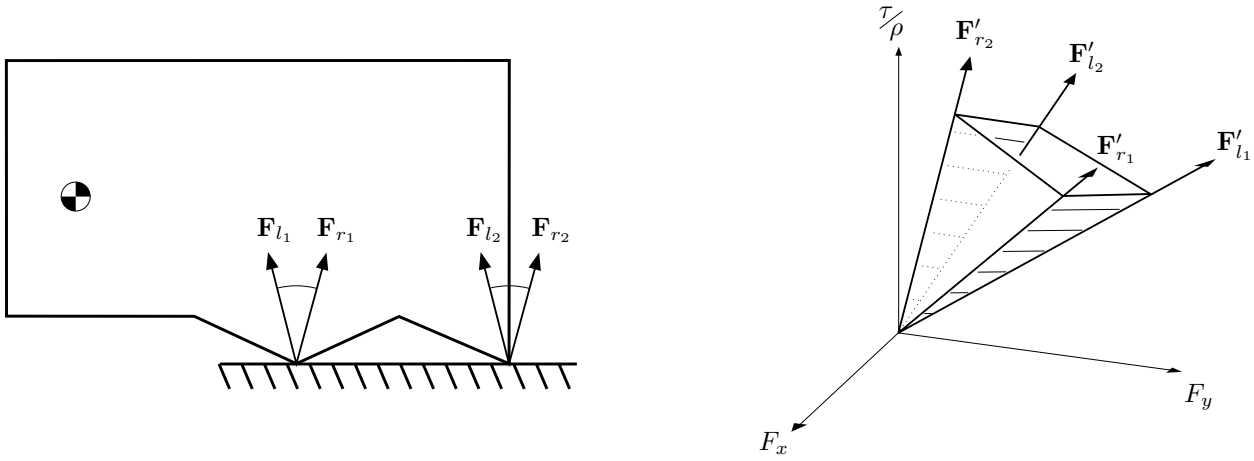


Figure 88: A multiple-contact configuration-space friction cone [Erdmann 1984].

Erdmann [1984] extended the friction cone construction to include these torques. By extending the (F_x, F_y) space with a third dimension for an applied torque τ, Erdmann was able to express the contact friction cone as a sector of an inclined plane in this space (Figure 87(b)). The elevation of each ray bounding this sector corresponds to the torque applied by the corresponding friction-cone ray. Because the linear and angular units of the force and torque axes are incompatible, Erdmann scaled the third axis by the polygon's radius of gyration ρ, producing an $(F_x, F_y, \tau/\rho)$ space of generalized forces. Because of the direct correlation between the contact normal expressed in the $(F_x, F_y, \tau/\rho)$ force space and the outward-pointing normal of c-surfaces expressed in the $(x, y, \rho\theta)$ configuration space, Erdmann referred to this construction as the *configuration-space friction cone.* We will adopt Erdmann's term, sometimes using *c-space friction cone* for brevity.

Erdmann also showed how to formulate configuration-space friction cones for multiple-contact situations. Since the total contact reaction force is a positive linear combination of the contact forces arising at individual contact points, the set of possible total reaction forces corresponds to all positive linear combinations of the individual c-space friction cones. This amounts to forming the convex hull of the c-space friction cones, as shown in Figure 88. This infinite polyhedral cone contains all of the contact reaction forces that can possibly be generated by the multiple-contact situation.

As in the particle example, static equilibrium is only possible when the applied force falls within the negated c-space friction cone. Otherwise, no contact force may be generated to exactly balance the applied force, and motion must occur. Thus the configuration-space friction cone gives us a direct method for determining whether static equilibrium is possible for an arbitrary contact situation: We simply form the negated c-space friction cone, and check whether the applied force ray is contained within the resulting infinite polyhedral cone.

This test may be simplified by projecting the three-dimensional force-torque space onto the surface of a unit sphere, which we will refer to as the *force-sphere* (Figure 89). This mapping preserves the validity of our possible-equilibrium test, but simplifies the required geometric analysis. For example, the three-dimensional infinite polyhedral cone becomes a convex polygon with great circle edges, and the required ray-in-polyhedral-cone test becomes a point-in-polygon test (Figure 90). This mapping first appeared in [Brost and Mason 1990], and was developed in a collaborative effort between Matt Mason and myself.

In addition to analyzing static equilibrium problems, configuration-space friction cones may also be used for full dynamic analysis and motion prediction. See [Erdmann 1984] for a complete review of dynamic analysis using configuration-space friction cones. Brost and Mason [1990] also address dynamic analysis, and show how Erdmann's analysis techniques may be simplified using two-dimensional constructions. Erdmann's results are also closely related to an independently-developed algebraic technique for analyzing multiple-contact friction problems [Rajan *et al.* 1987].

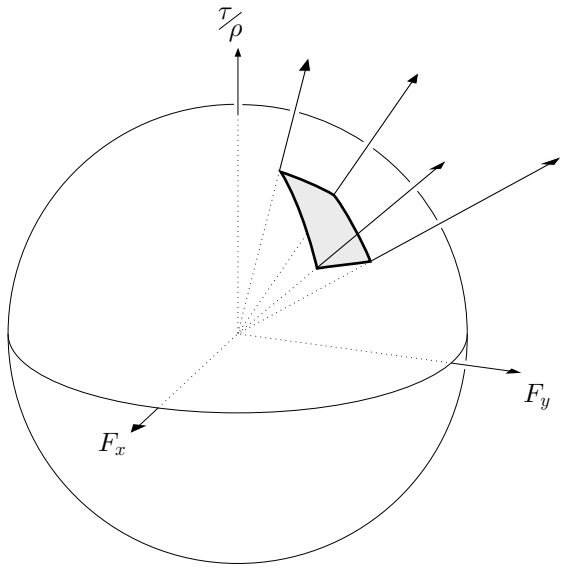


Figure 89: Projecting a configuration-space friction-cone onto the unit sphere [Brost and Mason 1990].

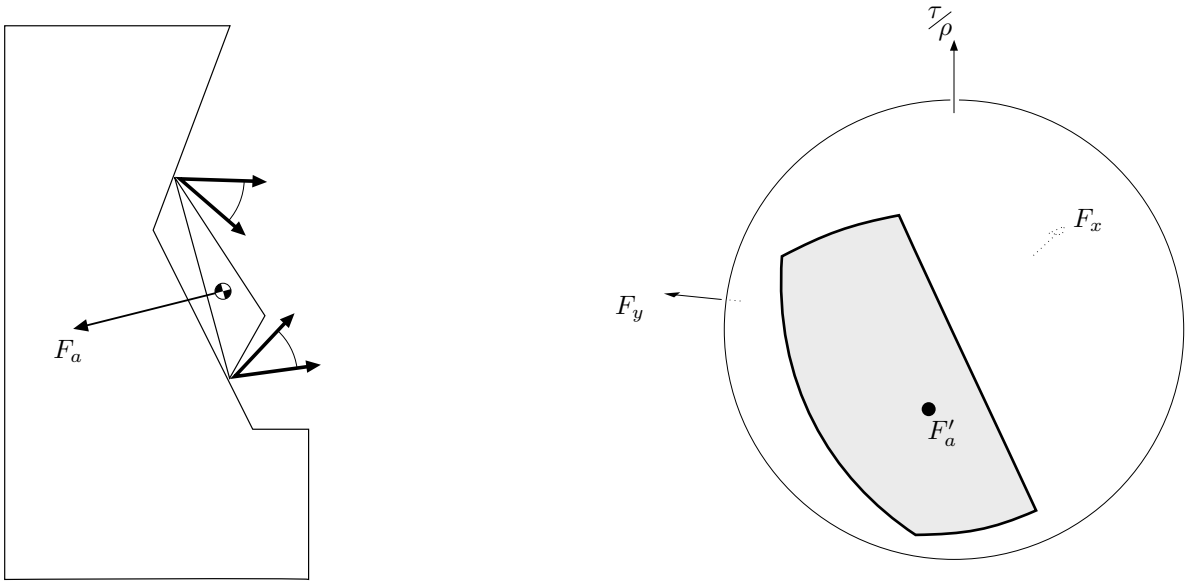


Figure 90: Determining whether static equilibrium is possible for a given contact configuration. The polygon on the sphere is the projection of the negated configuration-space friction cone; the highlighted point is the projection of the applied force. Equilibrium is possible if and only if the polygon contains the point.

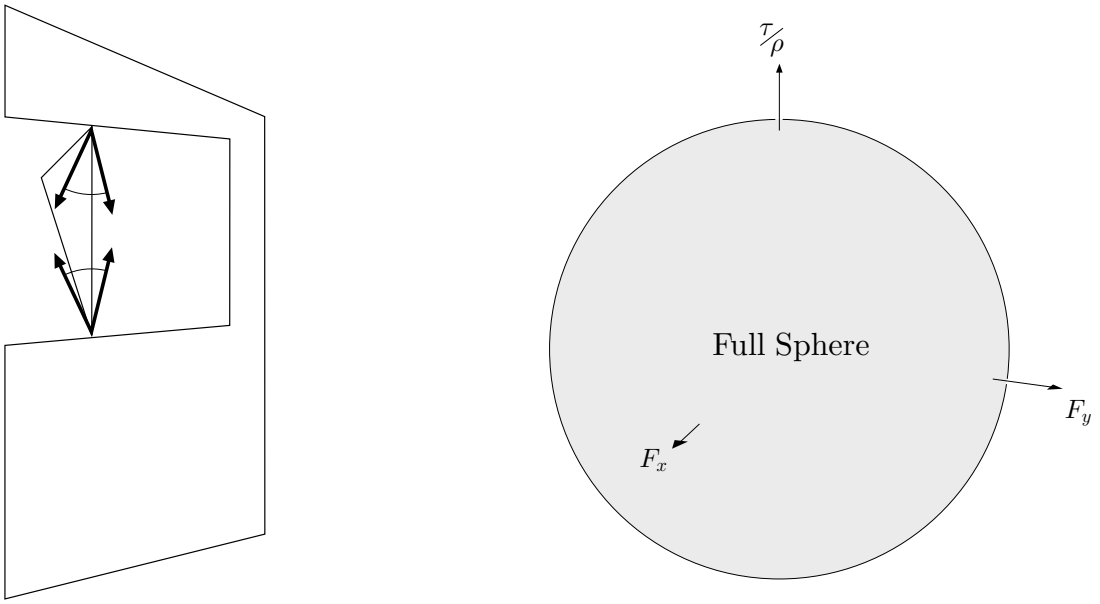


Figure 91: A force-closure contact condition, and the corresponding configuration-space friction cone.

Special Situations

Force-Closure

In some contact situations, static equilibrium is possible for any applied force. These situations correspond to *force-closure* conditions [Nguyen 1988], and are indicated whenever the configuration-space friction cone spans the entire force sphere. Figure 91 shows an example force-closure situation. In this situation, each contact friction cone contains the opposite friction cone's contact point. This condition allows aribtrary internal forces to develop in the polygon, thus producing arbitrarily large contact normal forces. These arbitrarily large contact normal forces can in turn produce arbitrarily large frictional reaction forces, and thus this contact situation may generate a reaction force to balance any finite applied force.

This example underscores the nature of our static-equilibrium analysis. When the applied force is contained in the negated configuration-space friction cone, then static equilibrium is possible; however, it is not guaranteed. To see why, imagine that two polygons have been placed in the configuration shown in Figure 91, and a 5-Newton leftward force is applied as shown in Figure 92(a). In this situation, static equilibrium is possible, but motion is also possible. If a large internal force is present in the triangle, then the contact reaction forces will be only slightly perturbed by the applied force, and static equilibrium will occur (Figure 92(b)). On the other hand, if little or no internal force is present, sufficient friction will not be available to balance the applied force, and the triangle will break contact (Figure 92(c)). These two outcomes cannot be distinguished without *a priori* knowledge of the triangle's internal stress, which is not represented in our rigid-body model of objects.

In order to assert that static equilibrium must occur, it is necessary to show that motion is not possible. This generally requires a full dynamic analysis, including an exhaustive test of all possible contact modes at each contact point. See [Erdmann 1984] or [Brost and Mason 1990] for details.

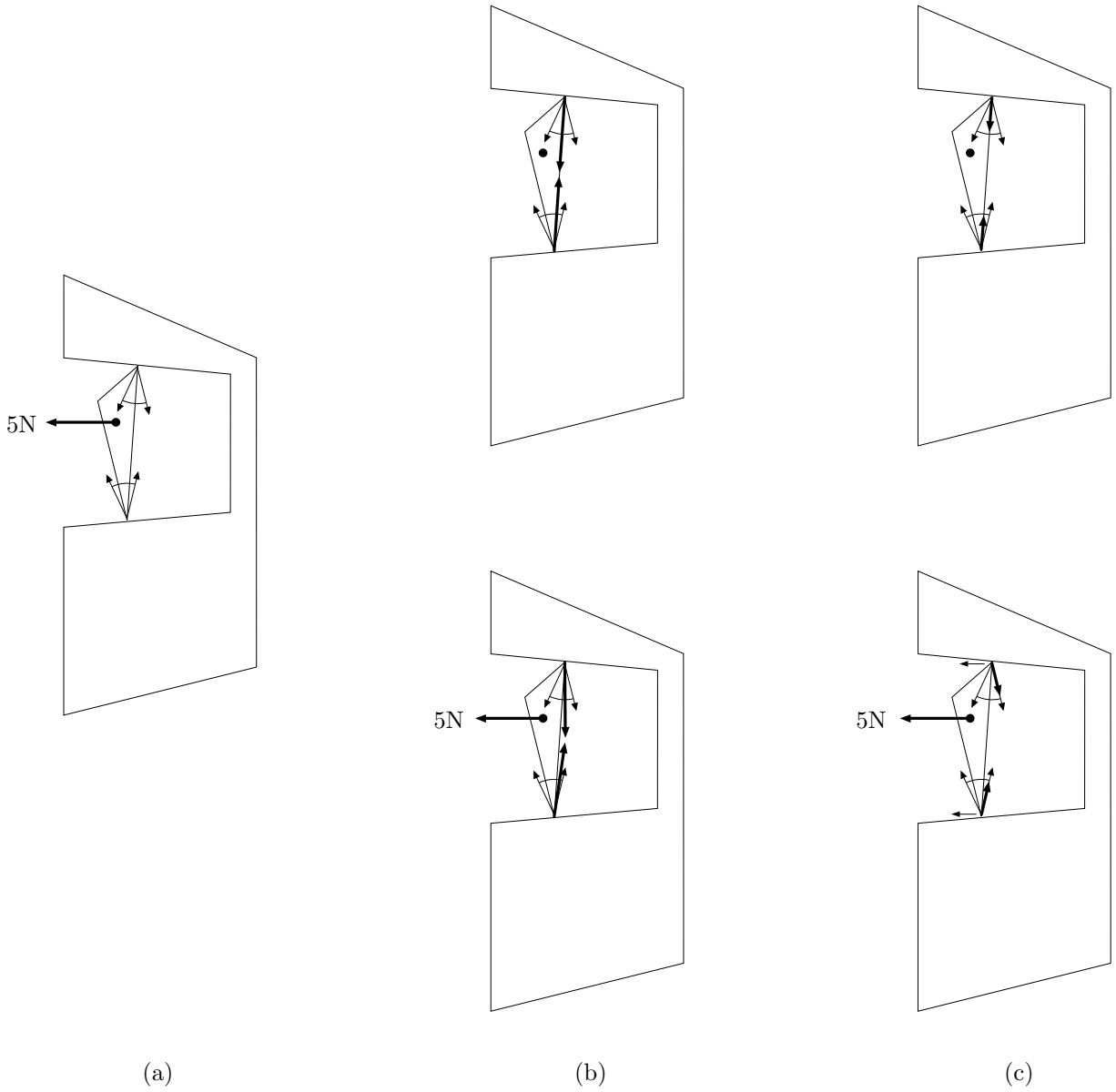


Figure 92: An ambiguity resulting from the rigid-body model of objects. (a) A force of 5 Newtons is applied to the triangle as shown. (b) The upper diagram shows a situation before the 5-Newton force is applied, where large contact forces exert internal stress on the triangle. The lower diagram shows the effect of applying the external force; the contact forces are perturbed slightly, but still balance the applied force. (c) A situation where the initial contact forces are small. These contact forces cannot balance the 5-Newton applied force, and the triangle slips away.

Adding Uncertainty

In most physical contacts, the coefficient of friction μ is not precisely known. We can include uncertainty in μ by constructing the configuration-space friction cone for both the minimum and maximum values of μ; this gives us the reliable and possible configuration-space friction cones, respectively. The reliable c-space friction cone corresponds to the set of contact reaction forces that can definitely be generated by the contact configuration, while the possible c-space friction cone corresponds to the set of forces that can possibly be generated by the contact configuration.

We may also include uncertainty in the applied force by expressing the applied force as a polygon of possible forces, instead of as a single known force. Under these assumptions, static equilibrium is possible whenever the applied force-polygon intersects the negated possible c-space friction cone. Similarly, static equilibrium will not reliably occur unless the applied force-polygon is completely contained in the negated reliable c-space friction cone. Because of the possiblity of motion ambiguities, this is a necessary but not sufficient condition for asserting that static equilibrium must occur. Figure 93 illustrates these constructions.

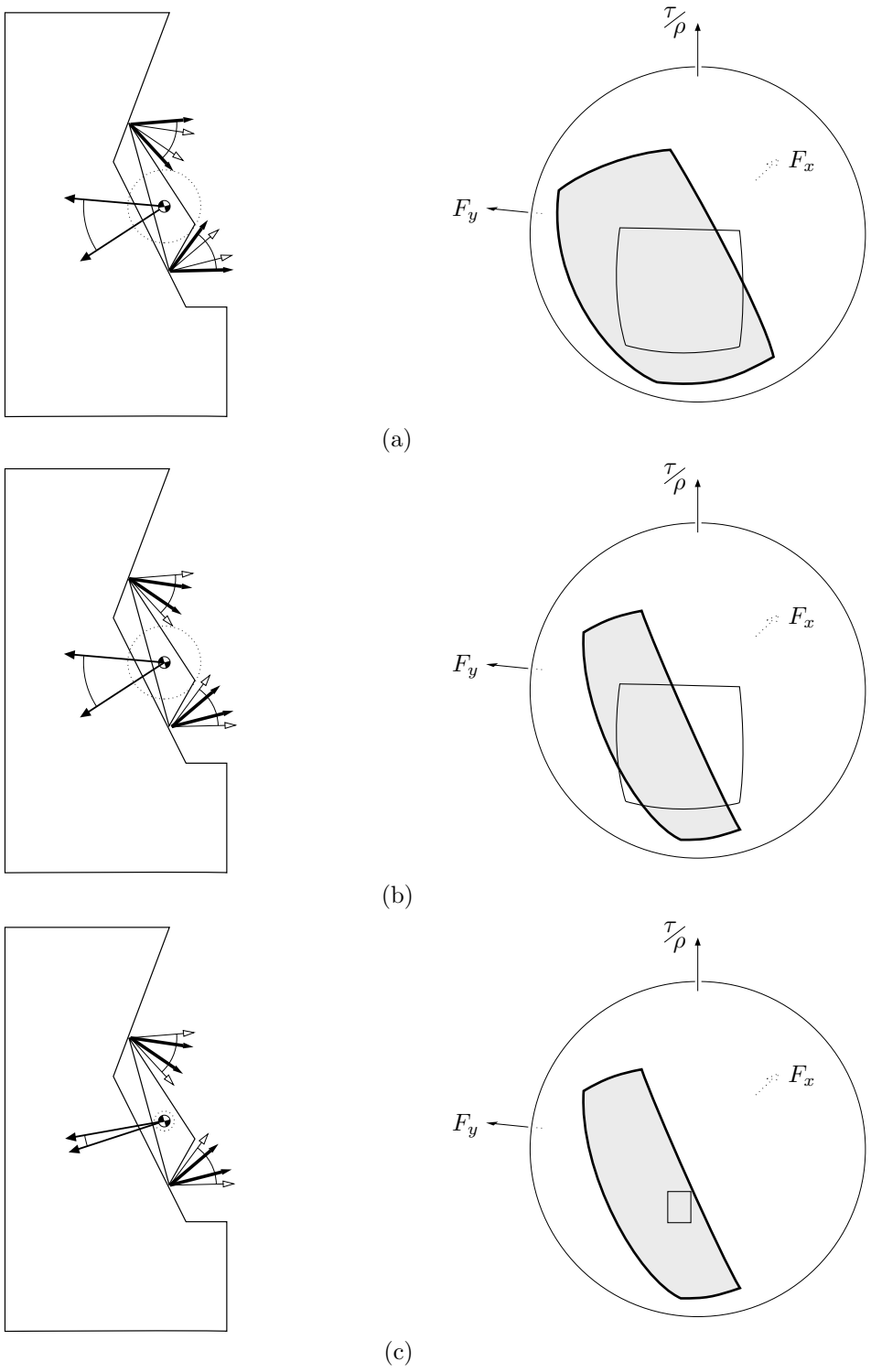


Figure 93: (a) The possible c-space friction cone, formed using $\mu_{\max}$. Since the polygon of possible applied forces intersects the possible c-space friction cone, static equilibrium is possible. (b) The reliable c-space friction cone, formed using $\mu_{\min}$. Since the polygon of possible applied forces is not fully contained within the reliable c-space friction cone, static equilibrium will not occur reliably. (c) The reliable c-space friction cone, with less uncertainty in the applied force. Since the applied-force polygon is fully contained within the reliable c-space friction cone, static equilibrium will occur reliably, as long as there are no motion ambiguities.

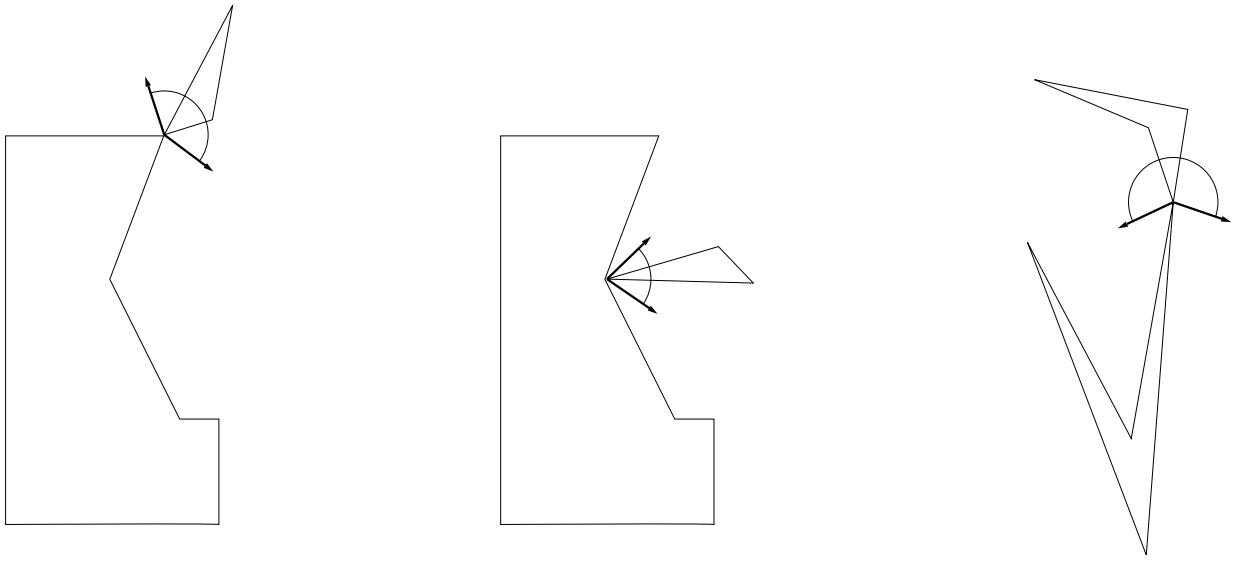


Figure 94: Example vertex-vertex contacts.

Vertex-Vertex Contacts

Constructing the c-space friction cone for a contact between two polygonal objects is generally straightforward; however, vertex-vertex contacts involve some subtleties. Figure 94 shows some example vertex-vertex contact conditions. Discerning a reasonable friction model for such contacts can be confusing. For example, for two very acute convex vertices in contact, what is the corresponding friction cone?

We can gain insight to this question by examining vertex-vertex contacts at a microscopic level. While our polygonal model of objects treats vertices as ideal points, real vertices of physical objects must have a finite radius of curvature. Figure 95 shows an example contact between two acute convex vertices. In the magnified views of this contact, it is apparent that several different contact normals are possible, depending on the microscopic relative positioning of the object vertices. Further, each of these contact normals has an associated friction cone, defined in the usual way.

These observations allow us to define the possible and reliable friction cones associated with a vertex-vertex contact. The possible friction cone is the set of contact reaction forces that can be generated by the contact, allowing any choice of microscopic contact geometry. The reliable contact friction cone is the set of contact reaction forces that can be generated by the contact, for every choice of microscopic contact geometry (Figure 96).

For pairs of convex vertices, these friction cones may be constructed as follows: First, construct the set of contact normals consistent with the vertex-vertex contact. This is accomplished by forming the set of outward-pointing-normals consistent with the fixed-vertex, and forming the set of inward-pointing normals consistent with the moving-vertex. These sets are then intersected to form the set of possible vertex-vertex contact normals. The possible contact friction cone is then constructed by growing this set by $\pm \arctan(\mu_{\max})$. The reliable contact friction cone is constructed by growing each endpoint of the contact-normal interval by $\pm \arctan(\mu_{\min})$, and intersecting the two resulting friction cones. Figure 97 illustrates these constructions.

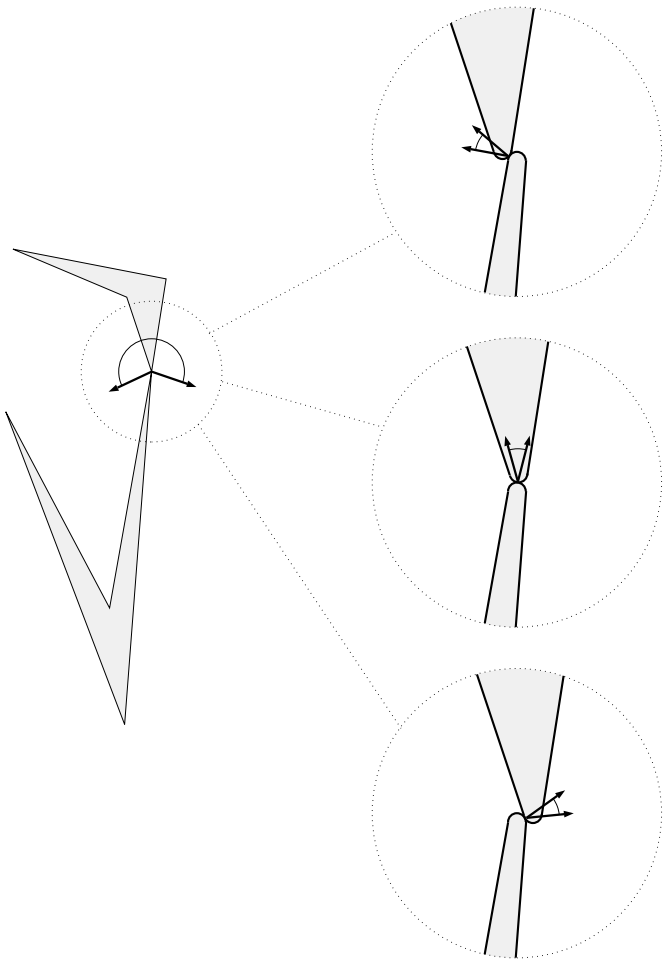
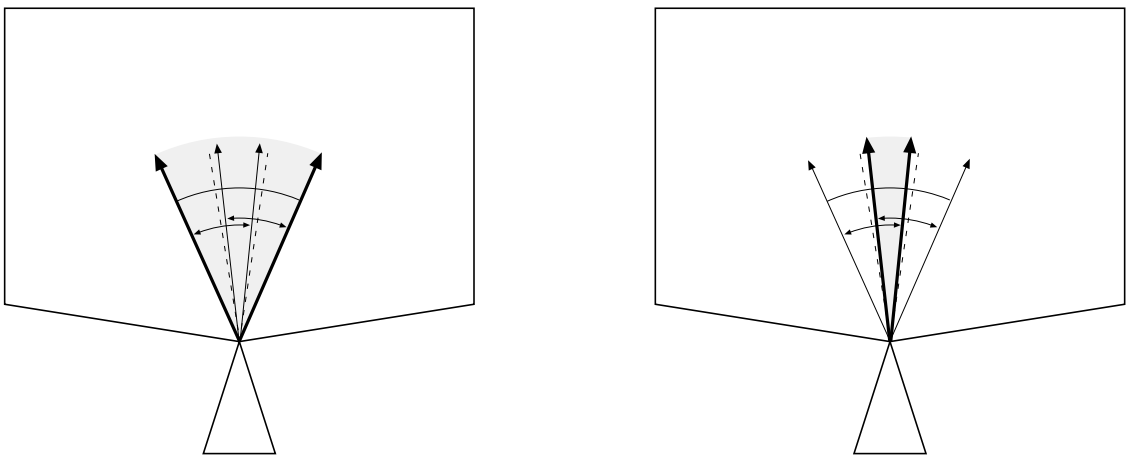
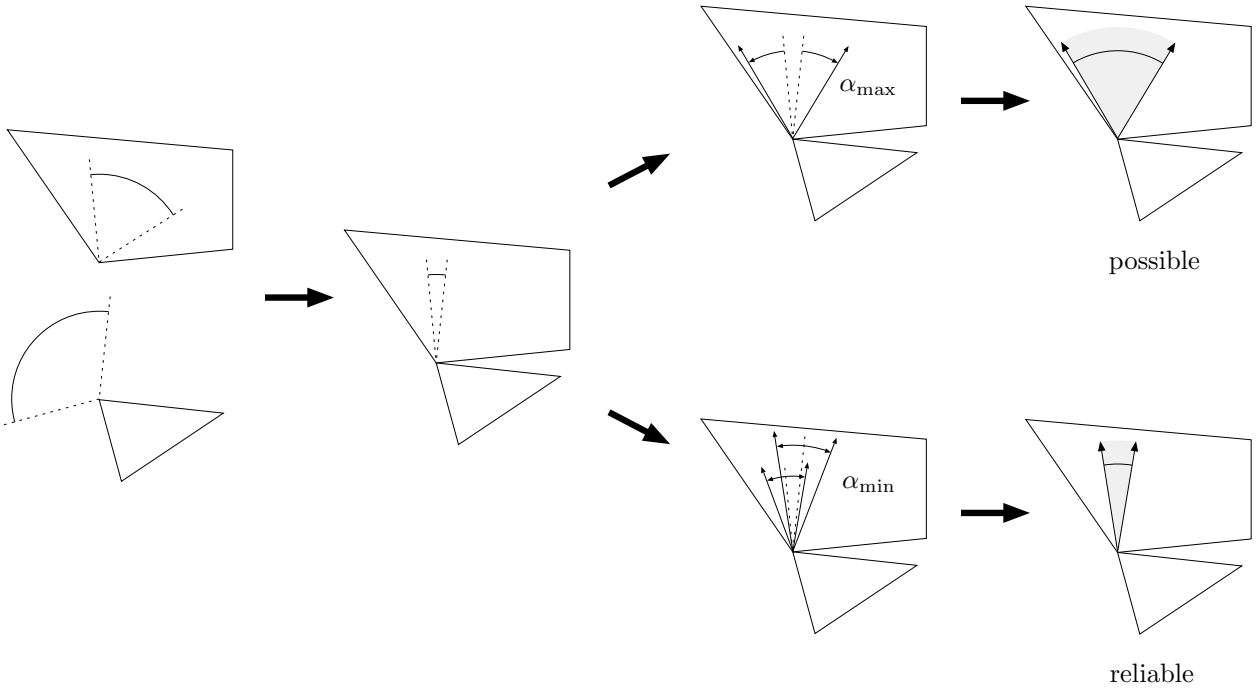


Figure 95: Magnified views of a vertex-vertex contact.

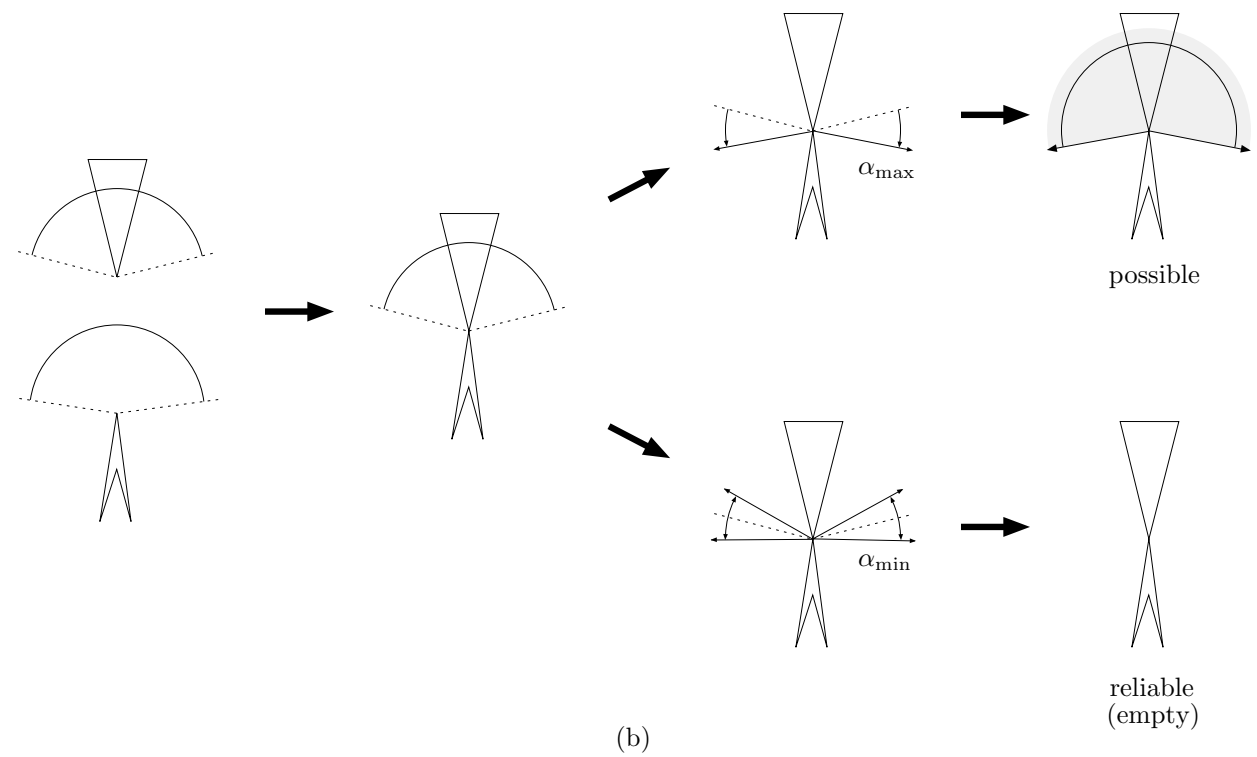


(a) (b)

Figure 96: The (a) possible and (b) reliable contact friction cones for a vertex-vertex contact. In this example, $\mu_{\min} = \mu_{\max}$.



(a)



(b)

Figure 97: Constructing the possible and reliable contact friction cones at a vertex-vertex contact. (a) and (b) show two examples. Here $\alpha_{\min} = \arctan(\mu_{\min})$, and $\alpha_{\max} = \arctan(\mu_{\max})$.

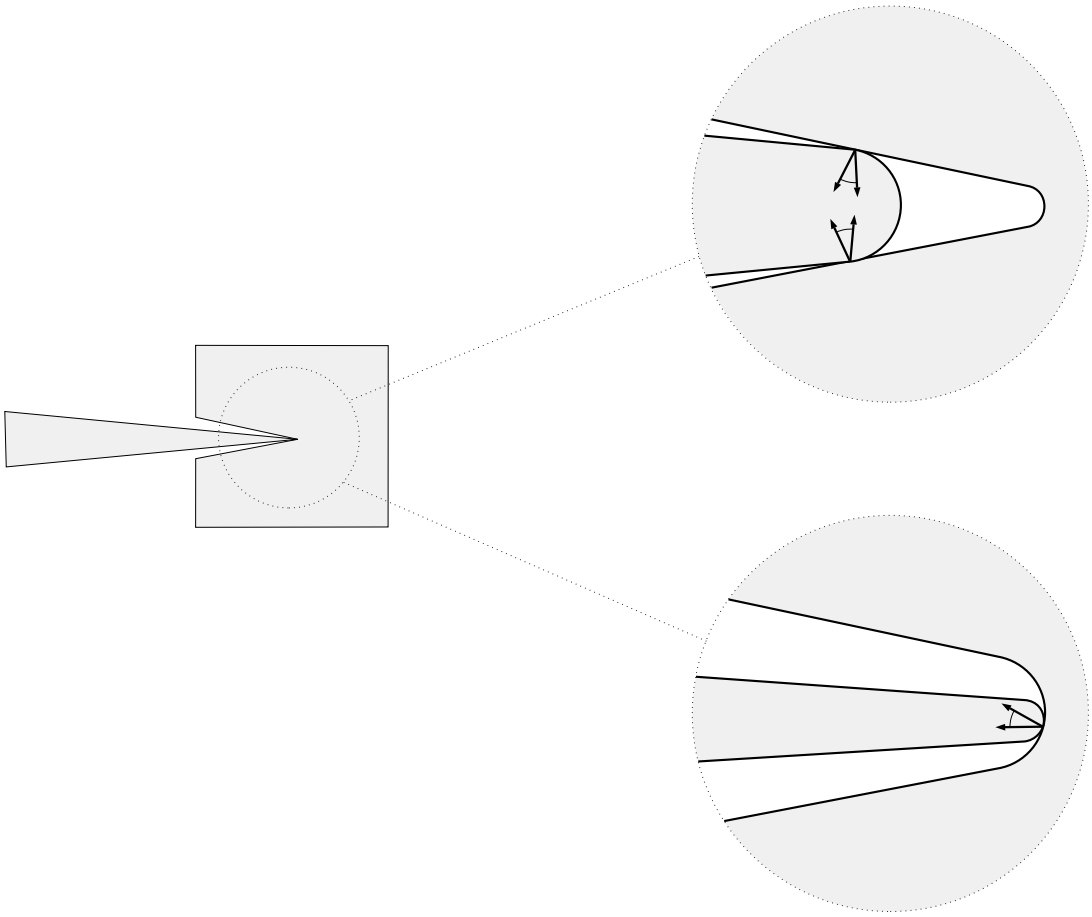


Figure 98: Magnified views of a concave vertex-vertex contact. In the upper view the concave vertex is sharper than the convex vertex, causing a microscopic force-closure condition. In the lower view the convex vertex is sharper, and microscopic force-closure does not occur.

Figure 97(b) illustrates an anomaly that can occur in the possible contact friction cone. Here the possible friction cone spans an interval of directions greater that 180°; convex-combinations of these forces will span all directions. Thus, if we apply the usual convex-hull operations on the force-sphere, the resulting c-space friction cone will predict that this contact can produce reaction forces that resist pulling the vertices apart — clearly an erroneous prediction.

A similar anomaly can arise if either vertex is concave, as shown in Figure 98. A microscopic view of this case reveals that it is possible for the contact to apply a reaction force in any direction, depending on the relative radius of curvature of the two vertices. If the concave vertex is sharper than the convex vertex, then a local force-closure contact condition may develop that is similar to the contact situation shown in Figure 91. If the convex vertex is sharper, then this local force-closure situation cannot occur.

We will resolve these anomalous situations by defining our model of possible and reliable friction cones to include these microscopic effects. For contacts involving convex vertices, we will apply the construction procedure defined above, and note that convex combination cannot be applied in cases where the resulting possible friction cone spans more than 180°. For contacts involving a concave

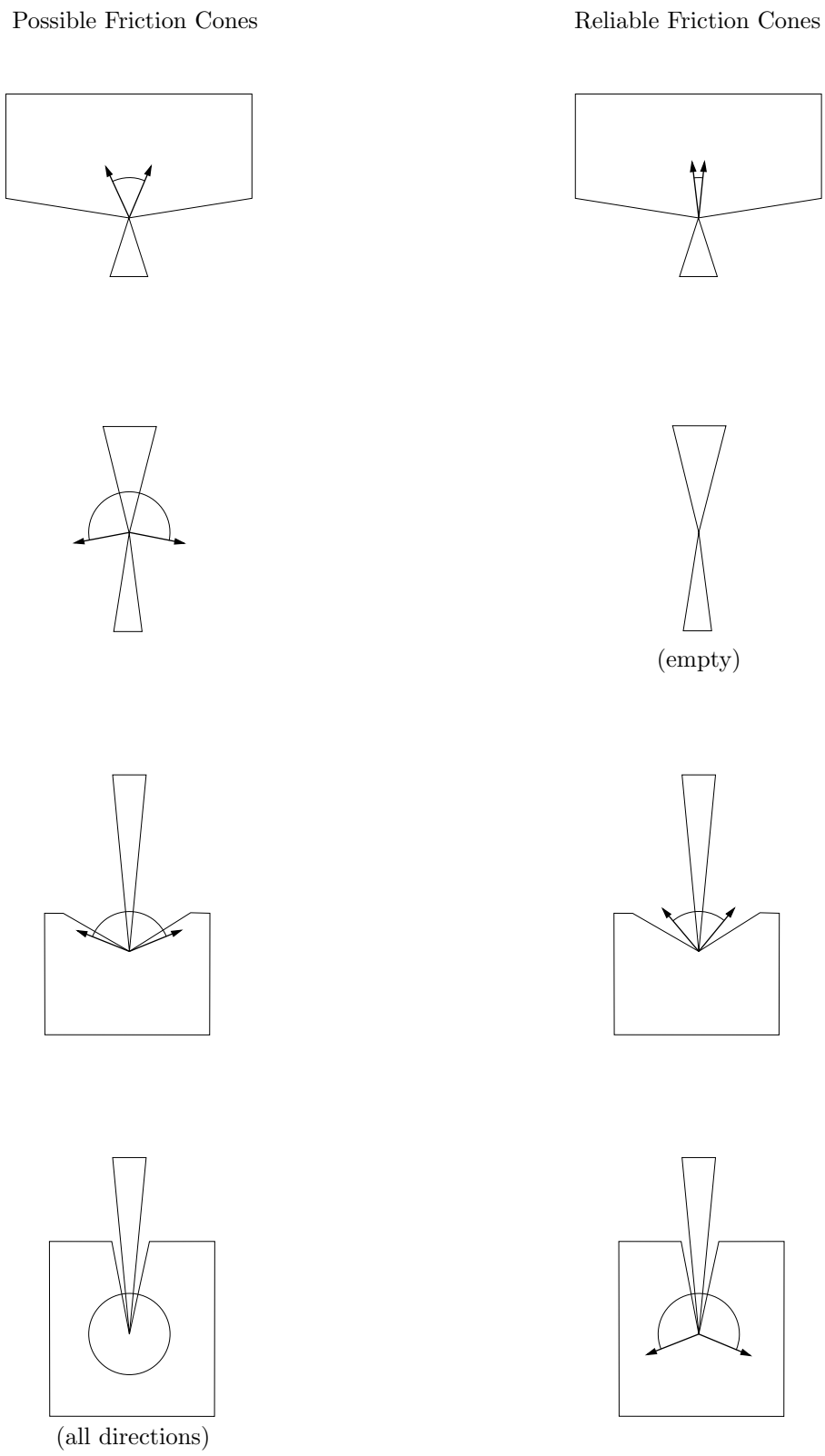
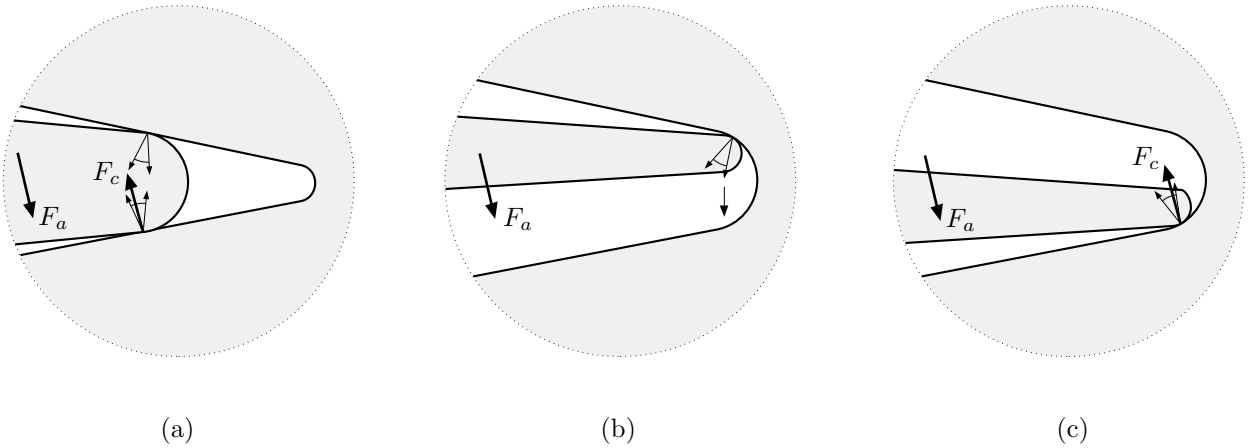


Figure 99: The interpretation of vertex-vertex contacts adopted in this thesis.



(a) (b) (c)

Figure 100: The microscopic displacement assumption. The microscopic contact in (a) can balance arbitrary forces instantaneously. The microscopic contact in (b) cannot instantaneously balance arbitrary forces, but will undergo a microscopic displacement to balance forces within the negated reliable friction cone. (c) An example microscopic displacement.

vertex, we will modify the construction procedure to (1) include all force directions if the possible friction cone spans more than 180°, and (2) to form the reliable friction cone by growing the interval of contact normals by $\pm \arctan(\mu_{\min})$, instead of applying the previous intersection procedure. If the resulting interval spans more than 180°, then we will add a note that convex-combination should not be applied. These modifications produce the interpretation of vertex-vertex contacts illustrated in Figure 99.

The construction of reliable friction cones for concave vertices requires further explanation. If the concave vertex is sharper than the convex vertex, then the convex vertex is in simultaneous contact with both edges adjacent to the concave vertex, and the contact can generate all forces within the reliable friction cone (Figure 100(a)). On the other hand, if the convex vertex is sharper than the concave vertex, then all forces within the reliable friction cone are *not* simultaneously available (Figure 100(b)). In light of this, why does the above construction procedure consider these forces to be reliably available? This seemingly contradictory conclusion is justified by assuming that applied forces within the reliable friction cone will either be countered by immediate contact reaction forces, or result in microscopic displacements that will eventually produce a countering reaction force (Figure 100(c)). This assumption is not necessarily justified by a full dynamic analysis, but is adopted because it yields a plausible macroscopic description of the contact forces that are reliably available at a concave vertex-vertex contact.

The interpretation shown in Figure 99 is one among several possible choices for modeling vertex-vertex friction cones. I have chosen this model because it provides a plausible macroscopic interpretation of the microscopic interactions that can occur at vertex-vertex contacts. For each force within a possible friction cone, there exists a microscopic contact condition that can produce the force. For each force within a reliable friction cone, every microscopic contact condition can produce the force, assuming that stabilizing infinitesimal displacements will occur. Moreover, as either vertex approaches a flat edge and $\mu_{\min} \to \mu_{\max}$, the possible and reliable friction cones both approach the ordinary friction cone originally developed by Moseley.

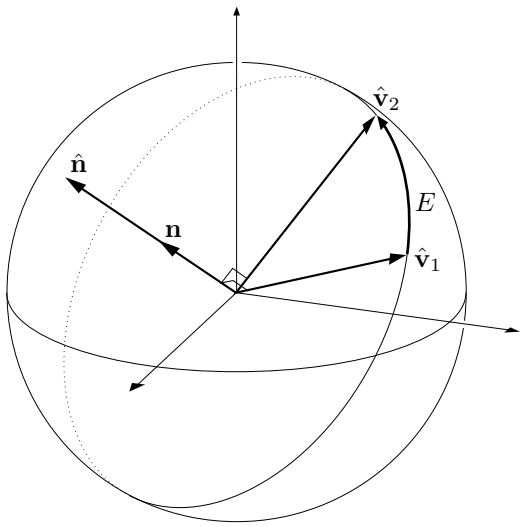


Figure 101: Representing a great-circle arc E with its endpoints $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$.

Representing Sets of Forces

The previous section presented Coulomb's model of dry friction, and described several geometric interpretations of this model. Among these interpretations were the configuration-space friction cone, and the possible and reliable contact friction cones. This section begins by developing a general data structure for representing polygons on the surface of a unit sphere, and then describes how to construct this data structure for the negated configuration-space friction cone. The section concludes by explaining how to construct a polygon of possible applied forces, given the applied-force description outlined in Figure 12 of Part I.

Representing Polygons on the Unit Sphere

The force sphere construction described above allows three-dimensional sets of forces to be represented by polygons on the surface of a unit sphere, with edges that are great-circle arcs. Often these polygons are convex; we will concentrate on this case first.

Points on the sphere may be represented with $\hat{\mathbf{v}} = [x \ y \ z]^T$ unit vectors. A pair of these points $(\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2)$ forms a great-circle arc on the sphere. The cross-product of these vectors is a vector $\mathbf{n}$, normal to the plane containing $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$; this vector implicitly represents the full great circle containing the arc (Figure 101). We will represent great-circle arcs with a triple $\langle \hat{\mathbf{v}}_1 \ \hat{\mathbf{v}}_2 \ \hat{\mathbf{n}} \rangle$, where $\hat{\mathbf{n}}$ is the result of normalizing $\mathbf{n}$.

Convex polygons on the sphere may be represented by a collection of edges, each of which is a great-circle arc. The endpoints of each edge are ordered so that their normal vectors point toward the polygon interior (Figure 102). This representation allows us to write a simple procedure for determining whether the polygon contains a given point $\hat{\mathbf{v}}$: The polygon contains $\hat{\mathbf{v}}$ if and only if $\hat{\mathbf{v}} \cdot \hat{\mathbf{n}}_i \geq 0$, for every polygon edge E_i. Determining whether two polygons intersect is also straightforward: The polygons intersect if and only if either polygon contains a vertex of the other polygon, or there exists a pair of edges selected from each polygon that intersect. The intersection point of a pair of edges may be constructed by computing the cross-product of their normal vectors, and normalizing and negating the result (Figure 103).

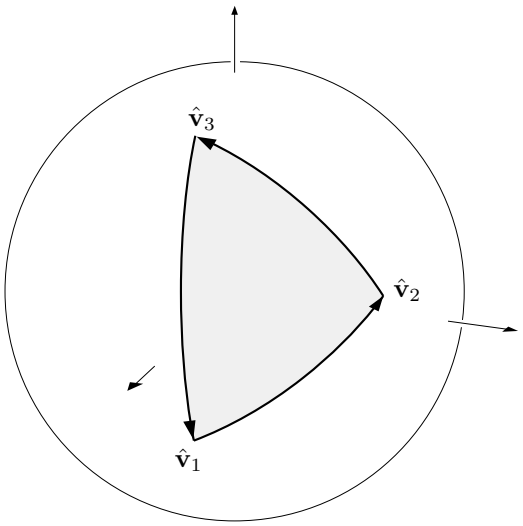
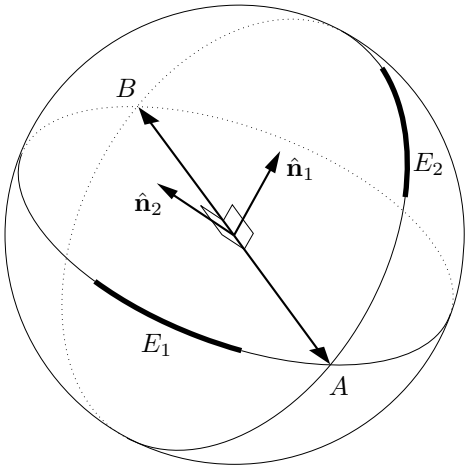


Figure 102: Representing a convex polygon on the sphere.



$$A = \mathit{normalize}\,(\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2)$$

$$B = -\mathit{normalize}\,(\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2)$$

Figure 103: Computing the intersection of two great-circle edges E_1 and E_2. We first compute the intersection points of the great-circles containing E_1 and E_2. These points are obtained by forming the cross-product of the edge normal vectors $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$; the resulting vector is normalized and negated to construct the two intersection points A and B. Point A is returned only if it lies between the endpoints of both E_1 and E_2; a similar test is applied to point B. Special code must be included to handle the case where E_1 and E_2 lie on the same great circle. The magnitudes of $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are reduced for clarity.

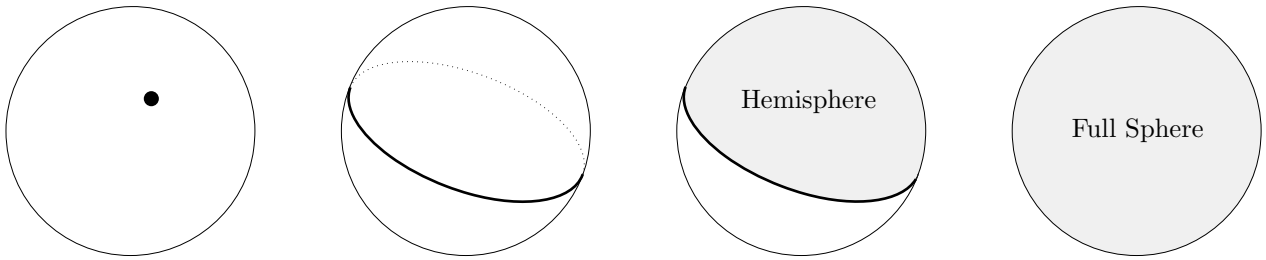


Figure 104: Special cases of convex polygons on the sphere.

convex-G-polygon

full-sphere?	A Boolean flag.
hemisphere?	A Boolean flag.
hemisphere-pole	A unit vector $[x\ y\ z]^T$.
full-great-circle?	A Boolean flag.
full-great-circle-normal	A unit vector $[x\ y\ z]^T$.
vertices	A set of unit vectors, containing the endpoints of each edge in the edges field. If this is a single-vertex polygon, then this set contains one vertex, while the edges field contains the null set.
edges	A set of G-arcs.

G-arc

$\hat{\mathbf{v}}_1$	Vertex 1, specified as an $[x\ y\ z]^T$ unit vector.
$\hat{\mathbf{v}}_2$	Vertex 2, specified as an $[x\ y\ z]^T$ unit vector.
$\hat{\mathbf{n}}$	The edge normal vector, determined by $\mathit{normalize}(\hat{\mathbf{v}}_1 \times \hat{\mathbf{v}}_2)$.

Figure 105: Data records for representing convex polygons on the unit sphere.

The geometry of the sphere allows special convex polygons that we will represent with special data structure fields. These include single-vertex polygons, full great-circles, arbitrary hemispheres, and the full sphere; these cases are illustrated in Figure 104. This expands our representation of convex polygons on the sphere to the data structure shown in Figure 105.

Extending this representation to include non-convex polygons on the sphere is straightforward: We simply form the union of several convex polygons. Thus our toplevel representation of a polygon on the sphere is a set of convex-G-polygon data records; convex polygons are represented by a set containing only one convex-G-polygon data record, while non-convex polygons are represented by a set containing several convex-G-polygon data records.

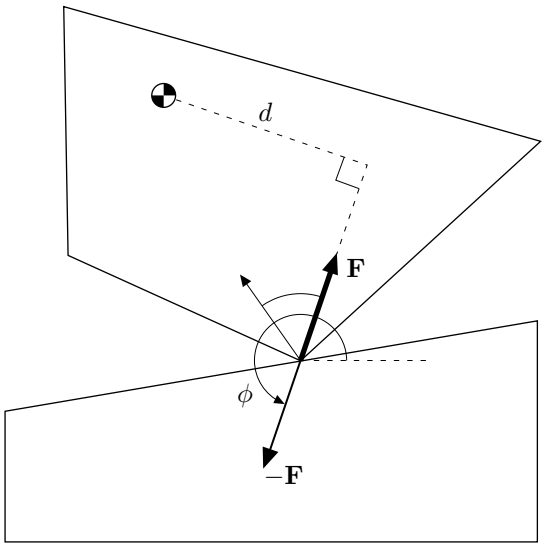


Figure 106: Parameters used in mapping a friction-cone ray **F** to its corresponding point on the force sphere. Since we are constructing the negated configuration-space friction cone, ϕ and d describe the negated ray $-\mathbf{F}$ rather than **F**. The parameter d is a signed quantity, and assumes a negative value when the reference point is on the opposite side of the ray $-\mathbf{F}$.

Representing Configuration-Space Friction Cones

The force-sphere representation of planar forces allows us to represent configuration-space friction cones using the spherical-polygon data structure described in the previous section. Given a contact configuration $(x, y, \theta, \text{contacts})$ between two polygonal objects and a table of friction coefficients $\mathcal{T}_\mu$, we can construct a polygon on the force sphere representing the negated possible or reliable configuration-space friction cone. This is accomplished by the following procedure:

1. Form the possible or reliable contact friction cone for every contact-pair, using the friction table $\mathcal{T}_\mu$ to look up the appropriate $\mu_{\min}$ or $\mu_{\max}$ friction coefficients.

2. Map the bounding rays of each contact friction cone into the $(F_x, F_y, \tau/\rho)$ force-torque space using the equation:

$$-\mathbf{F} = \begin{bmatrix} \cos(\phi) \\ \sin(\phi) \\ -\frac{d}{\rho} \end{bmatrix} \tag{1}$$

   where ϕ and d are defined in Figure 106. Here ρ denotes the radius of gyration of the moving-object; however, since we will not be using the configuration-space friction cone to perform dynamic analysis, any positive value of ρ may be chosen.

3. Normalize the resulting vectors and construct their convex hull, producing a spherical-polygon data structure. If one or more of the contact friction cones is attached to a note indicating that convex combinations should not be applied, split each such cone into two subcones, and form a convex-G-polygon for all combinations of these subcones with the other friction cones. The total negated configuration-space friction cone is then the union of these convex-G-polygons.

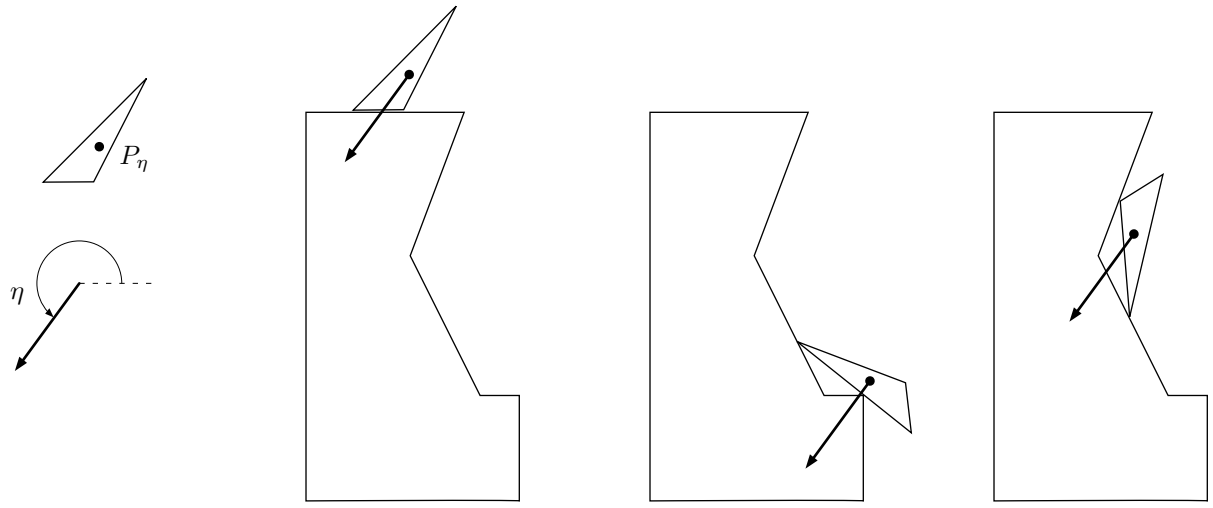


Figure 107: An example applied force that satisfies the criterion.

Representing the Set of Possible Applied Forces

The previous discussion has described a method for determining whether static equilibrium can occur, given a contact configuration and a polygon of possible applied forces. This method gives rise to the question: How do we convert a physical representation of an uncertain applied force into a polygon on the force sphere?

The answer to this question depends on the physical system producing the applied force. In the remainder of this chapter, we will restrict our attention to a particular class of physical systems; analysis of systems outside this class will be left to future work.

The systems we consider are characterized by an applied force that satisfies the following criterion: When the system is in static equilibrium, a nonzero relative applied force passes through a particular point P_η on the moving-object, in a constant direction η defined in the fixed-object's reference frame. Here "relative applied force" refers to the total force applied to the moving-object, minus the contact reaction force. This condition is illustrated in Figure 107.

While this criterion may seem abstract, it is satisfied by several physical systems. One example is an object falling in a uniform gravitational field; the criterion is satisfied if P_η is the object's center of mass, and η is the direction of the gravity vector. A second example is the compliant motion of an object moving under generalized damper control; here η is the nominal motion direction, and P_η is the control law's center of compliance. A third example is linear pushing, where P_η is the center of friction of the pushed object, and η opposes the pushing direction.

The criterion only considers forces that arise when the system is in static equilibrium; arbitrary forces are allowed at other times. This is emphasized by the linear pushing example, where the criterion is not satisfied except in situations where there is no relative motion between the pushing finger and pushed object. At all other times, the relative applied force is either zero or includes a moment that causes the line of force to pass to one side of the center of friction.

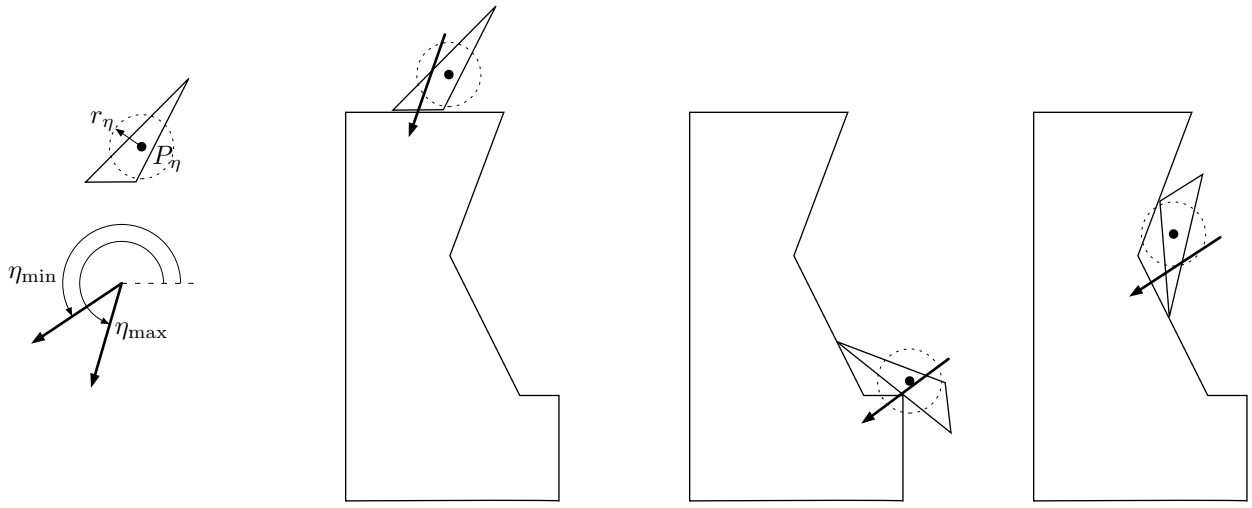


Figure 108: Modelling uncertainty in the relative applied force. The rays in the example configurations show typical applied forces that satisfy the uncertainty conditions.

One benefit of this criterion is that a force satisfying the criterion corresponds to a constant point on the force sphere, as long as the $(F_x, F_y, \tau/\rho)$ space is defined with respect to the point P_η. This requires that the moment d be measured relative to the moving-object's center of mass, center of compliance, or center of friction for the gravitational, compliant motion, and linear pushing examples. This represents a departure from Erdmann's standard of measuring all moments relative to the moving-object's center of mass, but this is not a problem for our purely static analysis.

We can include uncertainty in the applied force by treating the parameters P_η and η as uncertain variables. These modifications are illustrated in Figure 108, and lead to an extension of the applied-force criterion: When the system is in static equilibrium, a nonzero relative applied force passes through a point P'_η on the moving-object, in a direction η'. Here P'_η is a point inside a circle of radius r_η centered at a point P_η on the moving-object, and η' is in the interval $[\eta_{\min}, \eta_{\max}]$, defined in the fixed-object's reference frame. This criterion corresponds to the applied-force specification given for the *STATIC* algorithm in Figure 12.

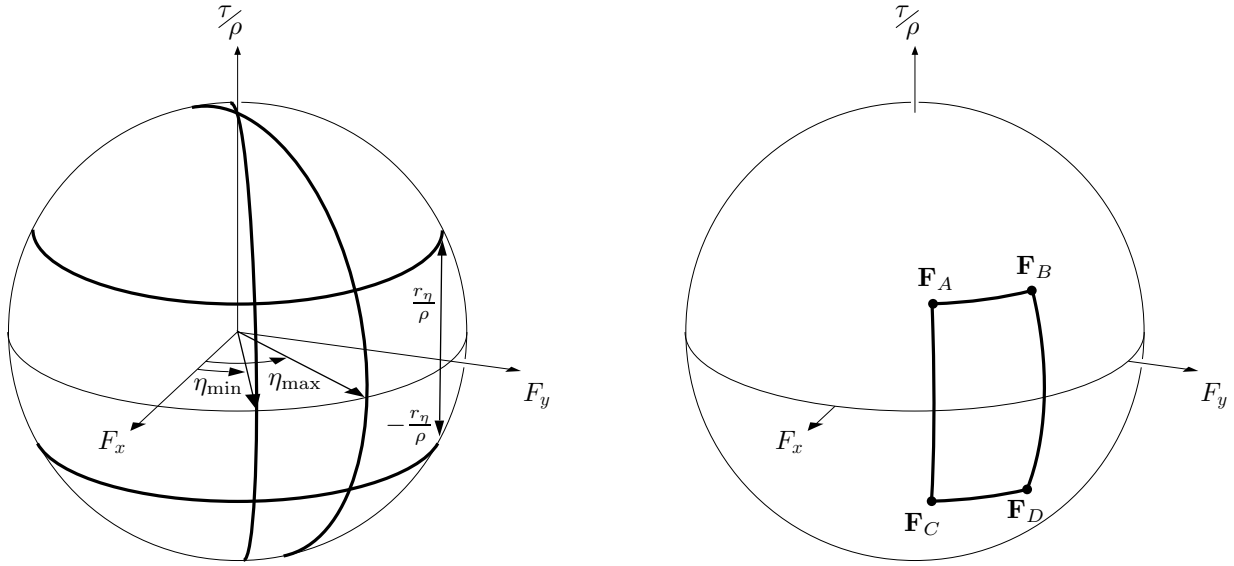
If the moving-object coordinate system is suitably defined, this criterion defines a set of forces that correspond to a constant polygon on the surface of the force sphere. This polygon is shown in Figure 109. Unlike the force-polygons seen previously, some edges of this polygon are not great circles; these edges correspond to latitude lines defining the maximum positive and negative moments that the applied force can exert with respect to P_η. As the error term r_η increases, these latitude lines move further from the force-sphere equator. The vertices of this polygon are given by the following equations:

$$\mathbf{F}_A = normalize \begin{bmatrix} \cos(\eta_{\min}) \\ \sin(\eta_{\min}) \\ \frac{r_\eta}{\rho} \end{bmatrix} \qquad \mathbf{F}_B = normalize \begin{bmatrix} \cos(\eta_{\max}) \\ \sin(\eta_{\max}) \\ \frac{r_\eta}{\rho} \end{bmatrix}$$

$$\mathbf{F}_C = normalize \begin{bmatrix} \cos(\eta_{\min}) \\ \sin(\eta_{\min}) \\ -\frac{r_\eta}{\rho} \end{bmatrix} \qquad \mathbf{F}_D = normalize \begin{bmatrix} \cos(\eta_{\max}) \\ \sin(\eta_{\max}) \\ -\frac{r_\eta}{\rho} \end{bmatrix}$$

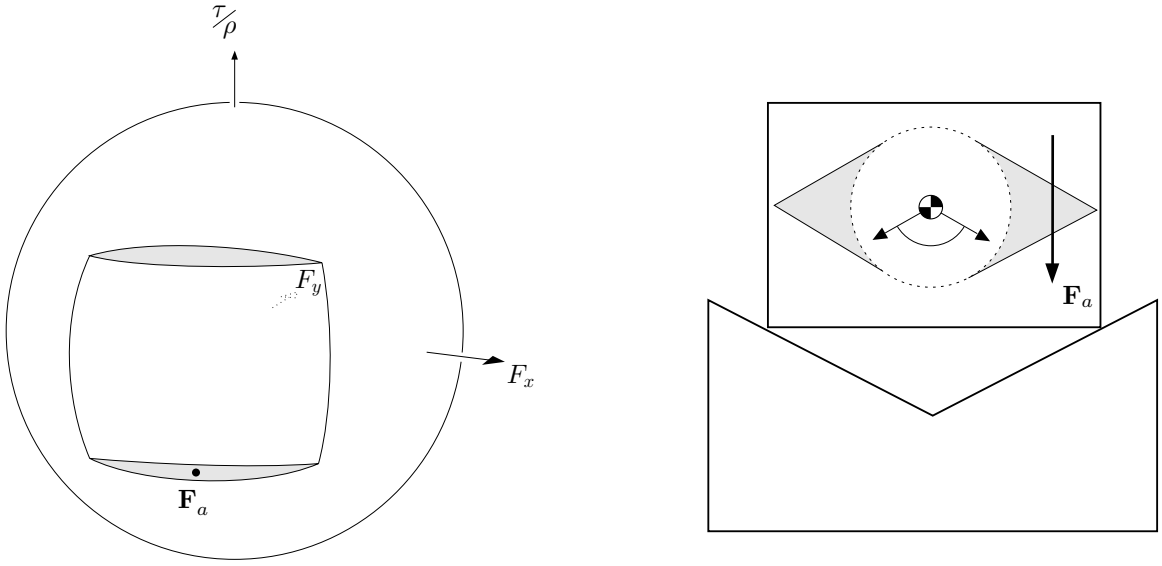
where ρ has the same value used in the configuration-space friction cone construction, and *normalize* indicates the vector-normalization operator.

We could represent this polygon exactly by extending our spherical polygon data structure to include latitude edges as well as great-circle edges, but instead we will construct a conservative approximation of the polygon using our existing data structure. This approximation is produced by using r_η and $[\eta_{\min}, \eta_{\max}]$ to construct the four polygon vertices, and then forming their convex hull. Figure 110 shows the result of this approximation; the discrepancy between the approximation and the true polygon increases as the applied force direction and moment errors increase. The implications of this approximation will be discussed at the end of this chapter.



(a) (b)

Figure 109: Converting the applied-force uncertainty defined in Figure 108 into a polygon on the force sphere. (a) The direction bounds $\eta_{\min}$ and $\eta_{\max}$ constrain the applied force to lie between two meridians on the sphere, while r_η gives rise to two latitude lines constraining the maximum positive and negative moments that can be exerted by the applied force. (b) The polygon of possible applied forces is the set of points on the sphere that satisfy all four constraints.



(a) (b)

Figure 110: The significance of our conservative approximation of the applied-force polygon. (a) The shaded region on the sphere shows the additional forces included by the conservative approximation. (b) The impact of the approximation in the original space. The approximation includes forces between $\eta_{\min}$ and $\eta_{\max}$ that pass through the shaded regions. $\mathbf{F}_a$ shows an example force that is included by the approximation, but not consistent with the uncertainty conditions.

Constructing Possible-Equilibrium Configurations

This section describes the *STATIC* algorithm, which constructs the set of configurations where static equilibrium is possible, given two polygonal objects $\mathcal{P}_m$ and $\mathcal{P}_f$, a table of friction coefficients $\mathcal{T}_\mu$, and a description of an applied force $\mathcal{F}_a$. Figure 111 shows an example set of possible-equilibrium configurations identified by the *STATIC* algorithm.

The *STATIC* algorithm uses the output of the *CO* algorithm as the basis for its analysis. Since equilibrium is possible only when the relative applied force is balanced by contact reaction forces, all possible-equilibrium configurations must correspond to contact configurations. This further implies that the set of possible-equilibrium configurations must be a subset of the configuration-obstacle surface.

The *STATIC* algorithm exploits this observation by *(i)* calling the *CO* algorithm to construct the configuration-obstacle surface, *(ii)* visiting each obstacle feature and constructing the possible-eqauilibrium subset of the feature, and *(iii)* returning the ensemble of the resulting subsets. As with the *CO* algorithm, the *STATIC* algorithm constructs its output incrementally, so that calling programs can avoid unnecessary computation.

The substance of the *STATIC* algorithm lies in the procedures used to construct the subset of each obstacle feature where static equilibrium is possible. A different approach is used to construct this subset for obstacle facets, edges, and vertices; these approaches are described in the following sections.

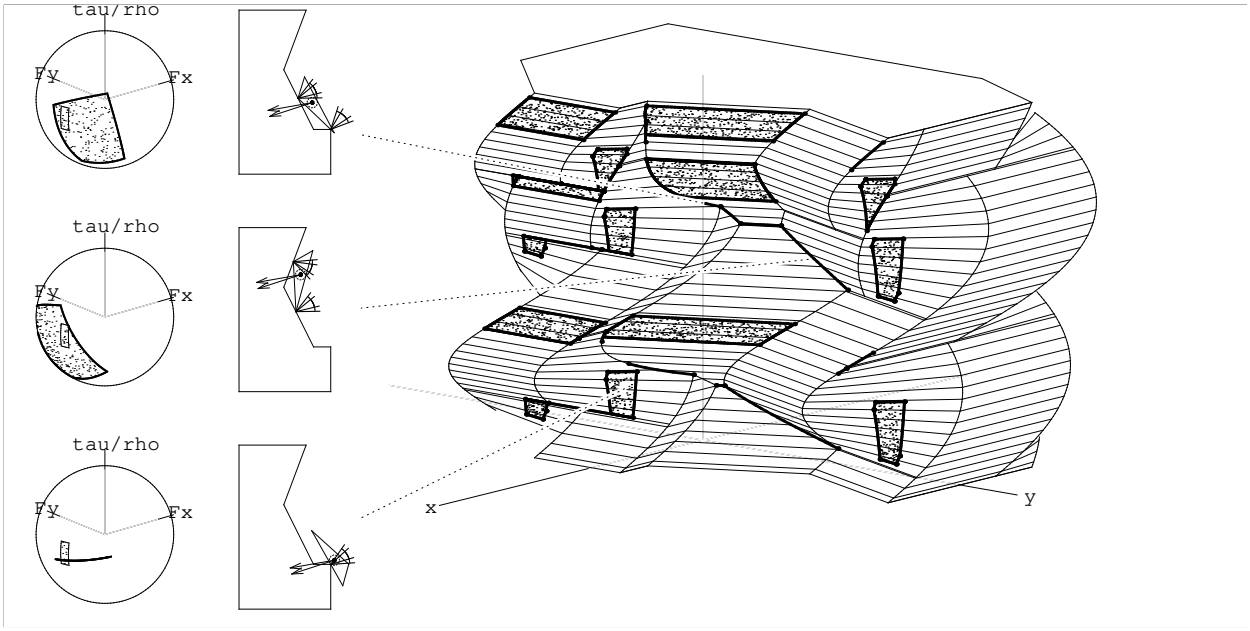


Figure 111: An example of the output generated by the *STATIC* algorithm. The highlighted features correspond to the set of contact configurations where static equilibrium is possible. In this example, $\mu \in [0.23, 0.48]$, $\eta \in [190°, 198°]$, and $r_\eta = 0.08l$, where l is the length of the longest triangle edge.

Obstacle Vertices

Since an obstacle vertex V is a single (x, y, θ) point, the subset of the vertex where static equilibrium is possible is either the point, or the null set. The *STATIC* algorithm determines whether or not equilibrium is possible for V by fetching V's contact set, constructing the negated configuration-space friction cone, and determining whether the polygon of possible applied forces intersects the configuration-space friction cone.

Obstacle Edges

Every obstacle edge is embedded in an unbounded curve in the configuration space; this curve is a one-dimensional set parameterized by p or θ, depending on whether the edge is class-0 or non-class-0. In the discussion that follows, we will only consider non-class-0 edges; the analysis of class-0 edges is analogous.

As the contact configuration varies along the unbounded curve, the configuration-space friction cone also varies. Figure 112 illustrates this variation for a typical ve-ve obstacle edge. Notice that the c-space friction cone and the applied-force polygon intersect for some configurations on the edge, but not for others; the *STATIC* algorithm is concerned with identifying the subset of the edge where intersection occurs.

This subset can be described by a set of intervals $\{[\theta_1, \theta_2], [\theta_3, \theta_4], \ldots\}$, where each θ_i corresponds to a configuration where the force-polygon and c-space friction cone transition from an intersecting to non-intersecting topological relationship. There are three types of these transitions, as shown in Figure 113.

The *STATIC* algorithm constructs the set of possible-equilibrium configurations on an edge by first identifying all of the θ-values that may give rise to a critical configuration on the force sphere, and then using these θ-values to assemble a collection of contiguous intervals where static equilibrium is possible. This computation is summarized in Figure 114.

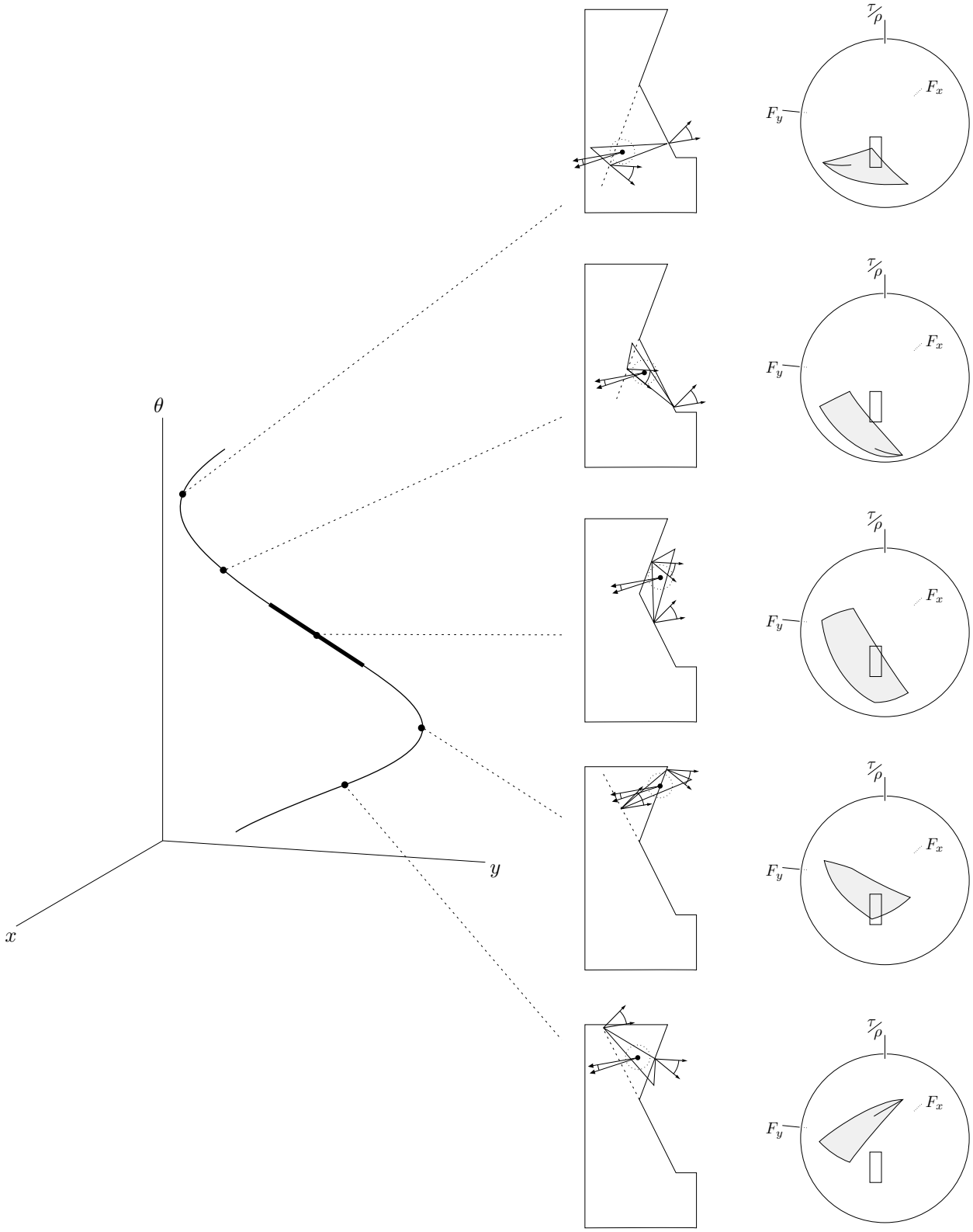


Figure 112: Variation of the negated c-space friction cone along an obstacle edge.

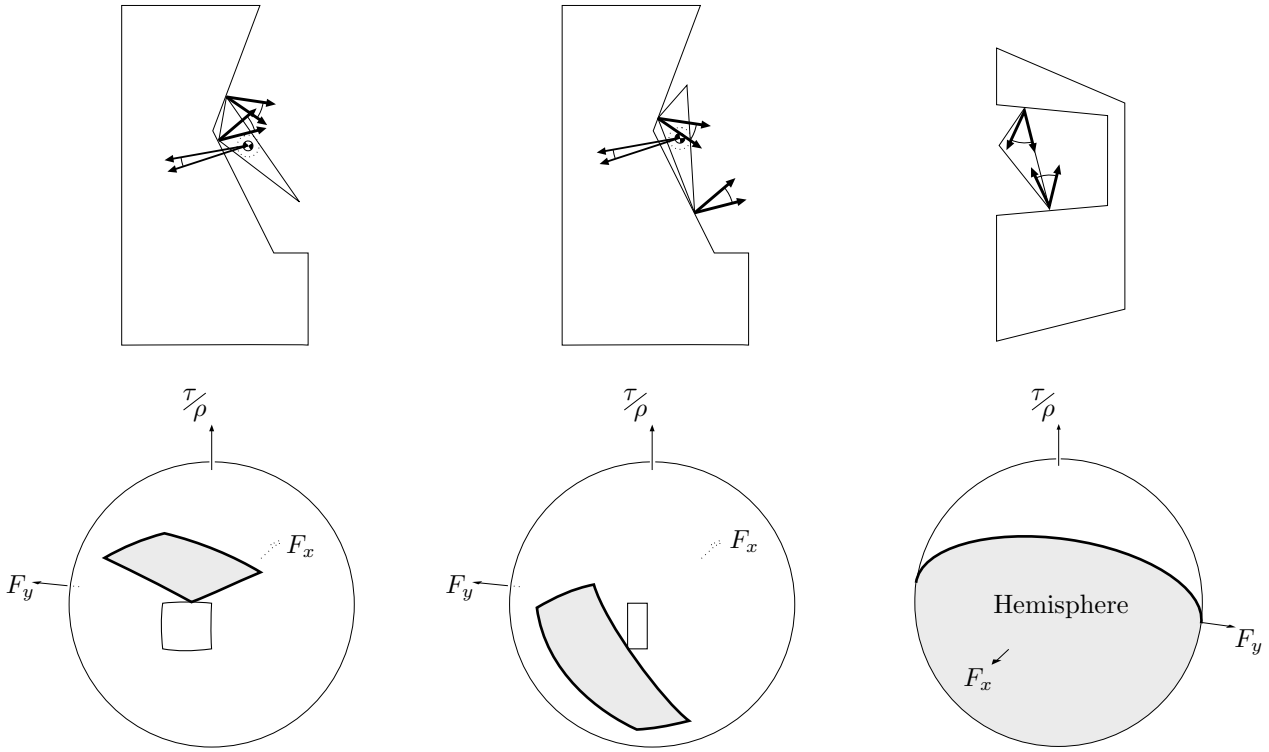
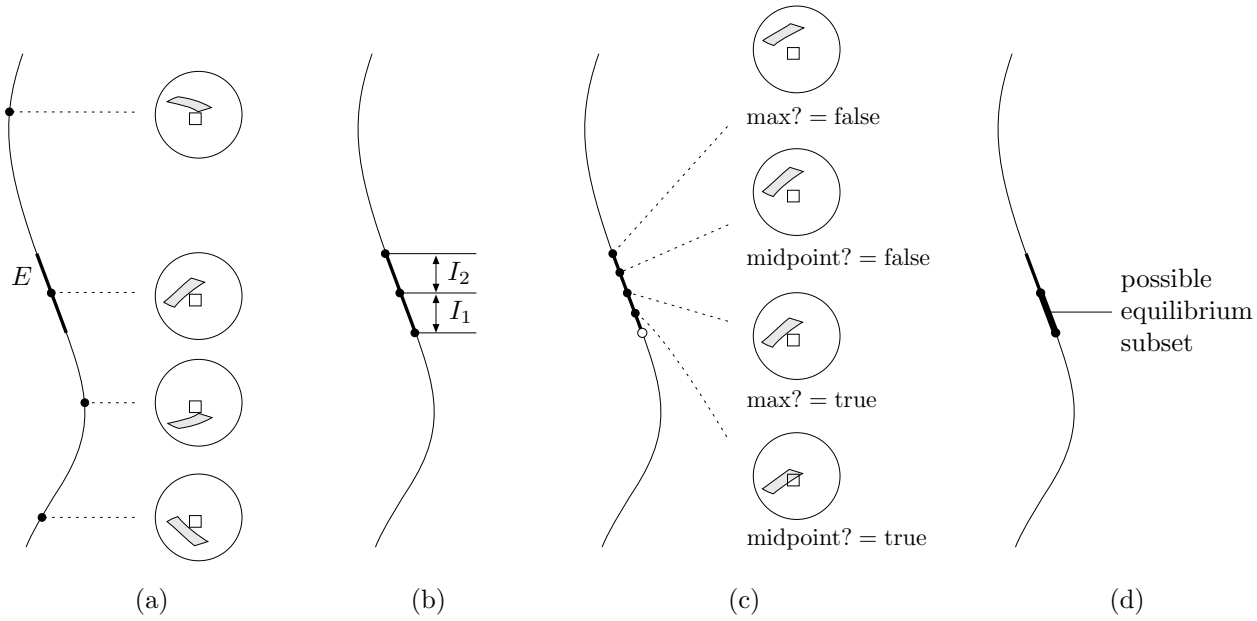


Figure 113: Critical configurations on the force sphere. The third configuration is a transition between a force-closure and a non-force-closure contact condition, and does not depend on the applied-force polygon.



construct-edge-possible-equilibrium-subset$_\theta(E, \mathcal{F}_a)$

Identify critical θ-values.
Using the critical-value analysis described in the text, identify the set of θ-values Θ_{critical} that correspond to possibly-critical configurations on the force sphere.

Assemble θ-intervals.
Discard all of the θ-values in Θ_{critical} that are not between the $(\theta_{\min}, \theta_{\max})$ endpoints of E. Sort the remaining θ-values to form a sequence of intervals $\mathcal{I} = [\theta_{\min}, \theta_1]\ \ [\theta_1, \theta_2]\ \ [\theta_2, \theta_3]\ \ \ldots\ [\theta_n, \theta_{\max}]$ that partition E.

Check the middle and maximum of each interval.
For each interval $[\theta_a, \theta_b] \in \mathcal{I}$, construct the negated c-space friction cone for the midpoint θ_{middle}, and determine whether or not it intersects the applied-force polygon $\mathcal{F}_a$. Apply a similar test to θ_b, and construct a record $\langle\, [\theta_a, \theta_b]$ midpoint? max? $\rangle$, where midpoint? and max? are set to true if an intersection occurred for θ_{middle} and θ_b, respectively. This process produces a sequence of records $\mathcal{I}_{\text{flags}} = \langle\, [\theta_{\min}, \theta_1]$ midpoint? max? $\rangle\ \ \langle\, [\theta_1, \theta_2]$ midpoint? max? $\rangle\ \ldots\ \langle\, [\theta_n, \theta_{\max}]$ midpoint? max? $\rangle$.

Form possible-equilibrium intervals.
Merge adjacent intervals in $\mathcal{I}_{\text{flags}}$ that have the same midpoint? field, forming the alternation $\mathcal{I}_{\text{merged}} = \langle\, [\theta_{\min}, \theta'_1]$ possible? $\rangle\ \ \langle\, [\theta'_1, \theta'_2]$ possible? $\rangle\ \ldots\ \langle\, [\theta'_n, \theta_{\max}]$ possible? $\rangle$, where the possible? flag of each record is the boolean complement of the preceding possible? flag. Include measure-zero intervals, which arise when the max? flag of an interval I_i is true, and the midpoint? flags of I_i and I_{i+1} are both false.

Return the subset of E where equilibrium is possible.
For every interval $I_i \in \mathcal{I}_{\text{merged}}$ with a possible? flag of true, construct a data record describing the subset of E defined by I_i. This will produce a c-edge data record for every finite interval where equilibrium is possible, and a c-vertex data record for every measure-zero interval where equilibrium is possible. Return this collection of data records.

(e)

Figure 114: Constructing the subset of an obstacle edge E where static equilibrium is possible. (a) Critical θ-values. (b) I_1 and I_2 are θ-intervals that partition E into two subintervals; configurations within each interval produce the same force-sphere topology. (c) The result of applying the possible-equilibrium test to the middle and maximum θ-values of I_1 and I_2. (d) The segment of E where equilibrium is possible. (e) The *construct-edge-possible-equilibrium-subset$_\theta$* procedure. This assumes that E is a non-class-0 edge; a similar procedure analyzes class-0 edges.

To identify the critical θ-values, we begin by observing that each critical condition illustrated in Figure 113 corresponds to a coincidence between a vertex and edge on the force sphere:

a. A vertex of the negated configuration-space friction cone is coincident with an edge of the applied-force polygon.

b. A vertex of the applied-force polygon is coincident with an edge of the negated configuration-space friction cone.

c. A vertex of the positive configuration-space friction cone is coincident with an edge of the negated configuration-space friction cone. This corresponds to the transition between a force-closure and non-force-closure configuration.

These coincidences can only occur when $\hat{\mathbf{v}} \cdot \hat{\mathbf{n}}_E = 0$, where $\hat{\mathbf{v}}$ is the force-sphere vertex, and $\hat{\mathbf{n}}_E$ is the normal vector of the force-sphere edge. Therefore these coincidence conditions may be expressed by the following three equations:

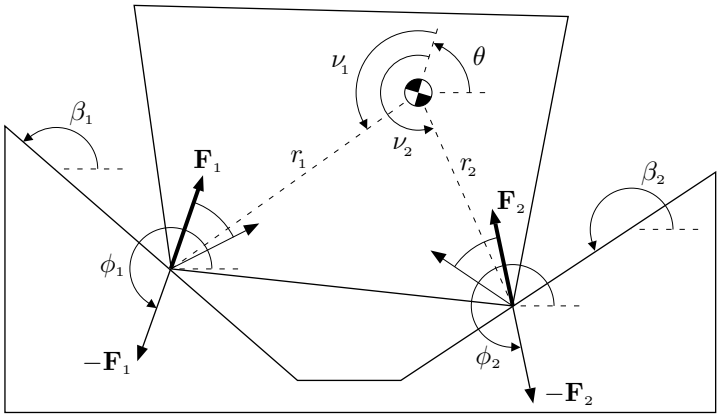
$$\begin{aligned} 0 &= -\mathbf{F}_{\mathrm{f}}(\theta) \cdot [\mathbf{F}_{\mathrm{a}1} \times \mathbf{F}_{\mathrm{a}2}] \\ 0 &= \mathbf{F}_{\mathrm{a}} \cdot [-\mathbf{F}_{\mathrm{f}1}(\theta) \times -\mathbf{F}_{\mathrm{f}2}(\theta)] \\ 0 &= \mathbf{F}_{\mathrm{f}1}(\theta) \cdot [-\mathbf{F}_{\mathrm{f}2}(\theta) \times -\mathbf{F}_{\mathrm{f}3}(\theta)] \end{aligned}$$

where $\mathbf{F}_{\mathrm{a}}$ is a vertex of the applied force-polygon, $-\mathbf{F}_{\mathrm{f}}(\theta)$ is a vertex of the negated configuration-space friction cone, and $\mathbf{F}_{\mathrm{f}}(\theta)$ is a vertex of the positive configuration-space friction cone. These equations can be developed for each type of obstacle edge by substituting the edge $f_x(\theta)$ and $f_y(\theta)$ functions appropriately and re-arranging terms to produce an equation of the form:

$$0 = k_0 + k_1 \mathrm{s}_\theta + k_2 \mathrm{c}_\theta + k_3 \mathrm{c}_\theta \mathrm{s}_\theta + k_4 \mathrm{s}_\theta^2 + k_5 \mathrm{c}_\theta^2 + k_6 \mathrm{c}_\theta \mathrm{s}_\theta^2 + k_7 \mathrm{c}_\theta^2 \mathrm{s}_\theta + k_8 \mathrm{s}_\theta^3 + k_9 \mathrm{c}_\theta^3$$

where s_θ and c_θ are abbreviations for $\sin(\theta)$ and $\cos(\theta)$, and each k_i is a constant that depends on the type of obstacle edge and the critical condition. Figure 115 shows an example equation derivation; a complete table of these equations is given in Appendix 4.

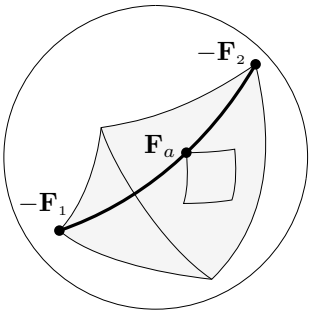
The *STATIC* algorithm uses these equations to identify the critical θ-values for an obstacle edge. Given an obstacle edge E and an applied-force polygon $\mathcal{F}_a$, the *STATIC* algorithm forms these equations for all appropriate combinations of edges and vertices of the corresponding c-space friction cone and applied-force polygon, and finds the roots of these equations using the numerical procedure described in Appendix 2. The resulting roots contain all possible critical θ-values, and are passed back to the *construct-edge-possible-equilibrium-subset*$_\theta$ procedure for interval construction and analysis.



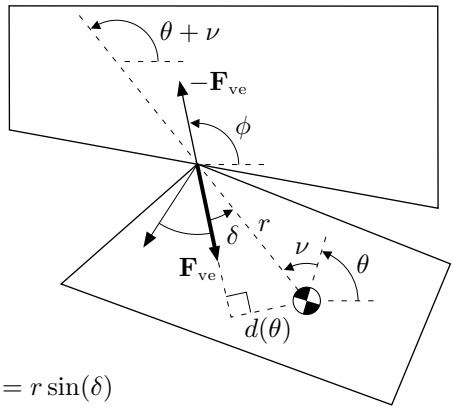
$$\phi_1 = \beta_1 + \tfrac{\pi}{2} + \arctan(\mu_1)$$

$$\phi_2 = \beta_2 + \tfrac{\pi}{2} - \arctan(\mu_2)$$

(a)



(b)



$d(\theta) = r\sin(\delta)$

$d(\theta) = r\sin(\theta + \nu - \phi)$

From equation (1) we have:

$$-\mathbf{F}_{\text{ve}}(\theta) = \begin{matrix} \mathrm{c}_\phi \\ \mathrm{s}_\phi \\ -\frac{d(\theta)}{\rho} \end{matrix}$$

Substituting $d(\theta)$ shown at left gives:

$$\boxed{-\mathbf{F}_{\text{ve}}(\theta) = \begin{matrix} \mathrm{c}_\phi \\ \mathrm{s}_\phi \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-\phi)} \end{matrix}}$$

(c)

Figure 115: Derviation of a critical-θ equation. (a) A ve-ve contact condition. The r, ν, β, and ϕ terms are constants; θ varies with configuration. (b) The critical condition we seek; a vertex of the applied-force polygon is coincident with an edge of the negated c-space friction cone. The friction-cone rays defining this edge are highlighted in (a). The vertices of the c-space friction cone define six possible connecting edges; all must be checked because the topology of the c-space friction cone may change with configuration. (c) An initial derivation step. The boxed equation maps a ve-contact friction-cone ray into the $(F_x, F_y, \tau/\rho)$ space. This equation exploits the fact that the ray's direction is constant, but the moment exerted by the ray varies with configuration.

$$
\begin{aligned}
0 &= \mathbf{F}_a \cdot [-\mathbf{F}_1(\theta) \times -\mathbf{F}_2(\theta)] \\
0 &= \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{\phi_1} \\ \mathrm{s}_{\phi_1} \\ -\frac{r_1}{\rho}\mathrm{s}_{(\theta+\nu_1-\phi_1)} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{\phi_2} \\ \mathrm{s}_{\phi_2} \\ -\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-\phi_2)} \end{bmatrix} \right) \\
0 &= \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \begin{bmatrix} \mathrm{s}_{\phi_1}\left(-\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-\phi_2)}\right) - \mathrm{s}_{\phi_2}\left(-\frac{r_1}{\rho}\mathrm{s}_{(\theta+\nu_1-\phi_1)}\right) \\ \mathrm{c}_{\phi_2}\left(-\frac{r_1}{\rho}\mathrm{s}_{(\theta+\nu_1-\phi_1)}\right) - \mathrm{c}_{\phi_1}\left(-\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-\phi_2)}\right) \\ \mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1} \end{bmatrix} \\
0 &= \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \begin{bmatrix} \mathrm{c}_\theta\left[\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)} - \frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)}\right] + \mathrm{s}_\theta\left[\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)} - \frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)}\right] \\ \mathrm{c}_\theta\left[\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} - \frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)}\right] + \mathrm{s}_\theta\left[\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} - \frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)}\right] \\ \mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1} \end{bmatrix} \\
0 &= x_a \cdot \left\{\mathrm{c}_\theta\left[\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)} - \frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)}\right] + \mathrm{s}_\theta\left[\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)} - \frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)}\right]\right\} \\
&\quad + y_a \cdot \left\{\mathrm{c}_\theta\left[\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} - \frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)}\right] + \mathrm{s}_\theta\left[\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} - \frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)}\right]\right\} \\
&\quad + z_a \cdot \left\{\mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1}\right\} \\
0 &= \mathrm{c}_\theta \cdot \left[x_a\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)} - x_a\frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} + y_a\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} - y_a\frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)}\right] \\
&\quad + \mathrm{s}_\theta \cdot \left[x_a\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)} - x_a\frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} + y_a\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} - y_a\frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)}\right] \\
&\quad + \quad [z_a\left(\mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1}\right)]
\end{aligned}
$$

Thus

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

where

$$k_0 = z_a\left(\mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1}\right)$$

$$k_1 = x_a\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)} - x_a\frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} + y_a\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} - y_a\frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)}$$

$$k_2 = x_a\frac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)} - x_a\frac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} + y_a\frac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} - y_a\frac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)}$$

(d)

Figure 115 (continued): (d) Derivation of the critical-θ equation. The θ-roots of this equation correspond to all configurations on the obstacle edge where the force-polygon vertex $\mathbf{F}_a$ lies on the great-circle defined by $-\mathbf{F}_1$ and $-\mathbf{F}_2$.

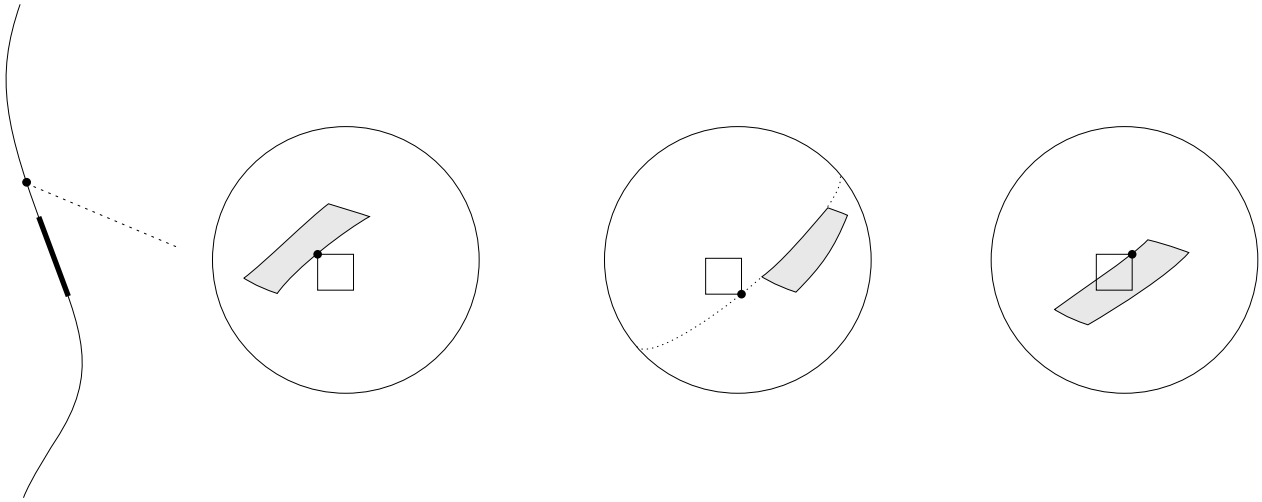


Figure 116: Non-critical force-sphere configurations.

Optimizations

The above technique always identifies all θ-values that give rise to critical conditions on the force sphere, but it also identifies additional non-critical θ-values (Figure 116). These values are discarded during later interval merging operations, but the algorithm's performance can be improved by discarding non-critical θ-values early on. This is accomplished by applying the following three tests to each θ-value as it is constructed:

a. If the θ-value is outside the $[\theta_{\min}, \theta_{\max}]$ limits of the obstacle edge, it is discarded immediately.

b. The θ-value is used to construct the force-sphere edge and vertex being analyzed. If the vertex lies outside the edge endpoints, the θ-value is discarded.

c. The θ-value is used to construct the force-sphere edge being analyzed and all vertices of the applied-force polygon and the c-space friction cone. The θ-value is discarded if any of the following conditions are violated:

   1. All applied-force polygon vertices are on the same side of the edge.
   2. All c-space friction cone vertices are on the same side of the edge.
   3. The applied-force polygon vertices and c-space friction cone vertices are on opposite sides of the edge.

   These conditions ensure that the edge/vertex coincidence condition separates the applied-force polygon and the c-space friction cone into disjoint half-spaces. If these conditions are not satisfied, the θ-value is not critical, and it is discarded.

These tests eliminate the non-critical θ-values illustrated in Figure 116 (a), (b), and (c), respectively.

The performance of the algorithm can also be improved by realizing that certain critical conditions can never occur. For example, a force-closure critical condition cannot occur for a ve-ve or ev-ev obstacle edge if the contact friction cone directions for one edge do not overlap the negated contact friction cone directions for the other edge. Similarly, static equilibrium is never possible for a ve-ve obstacle edge if the applied force-direction interval $[\eta_{\min}, \eta_{\max}]$ does not overlap the convex span of the negated contact friction-cone directions for both fixed-polygon edges. The *STATIC* algorithm includes these and other tests to avoid unnecessary critical-value analysis.

Obstacle Facets

Facets of the configuration-space obstacle correspond to single-point contact configurations. Because of the simplicity of these contacts, we can construct the set of possible-equilibrium configurations directly.

Static equilibrium with a single-point contact is only possible when the line of applied force passes through the contact point, in a direction within the negated possible contact friction cone. These conditions implicitly define a region of the facet c-surface where equilibrium is possible; the *STATIC* algorithm constructs this region, and then intersects it with the facet to construct the set of reachable possible-equilibrium configurations. This process is illustrated in Figures 117 and 118.

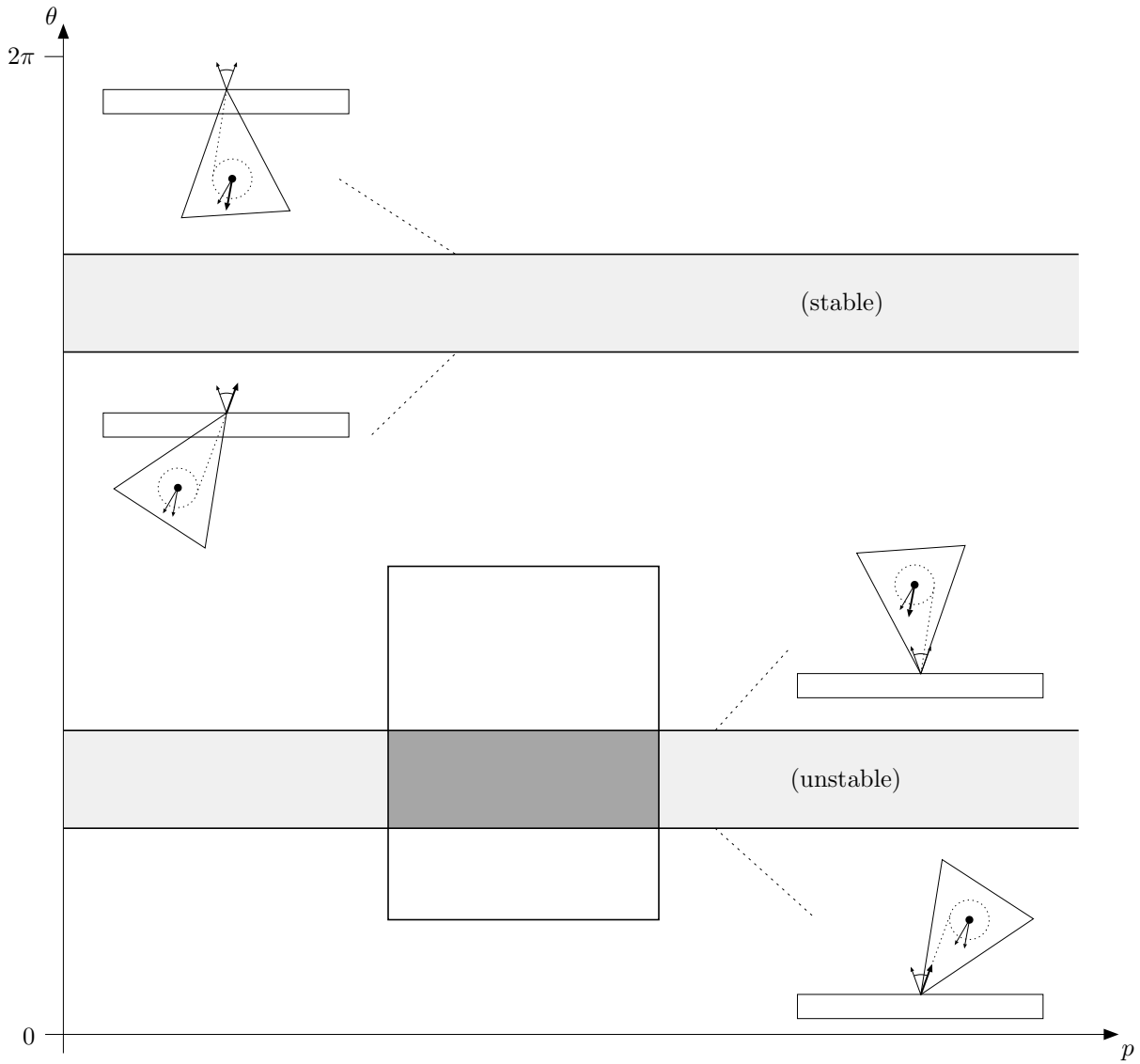


Figure 117: Identifying the possible-equilibrium configurations for a ve facet. The possible-equilibrium subset of the facet c-surface is a pair of infinite horizontal bands; these bands are delineated by critical θ-values corresponding to conditions where only one possible line of force can give rise to static equilibrium. These critical conditions are labelled, and the critical ray is highlighted. These bands do not always appear on the infinite c-surface; if $[\eta_{\min}, \eta_{\max}]$ does not overlap the negated contact friction cone, the bands vanish. Once these bands are constructed on the infinite c-surface, they are intersected with the facet to identify the set of reachable possible-equilibrium configurations (shown heavily shaded).

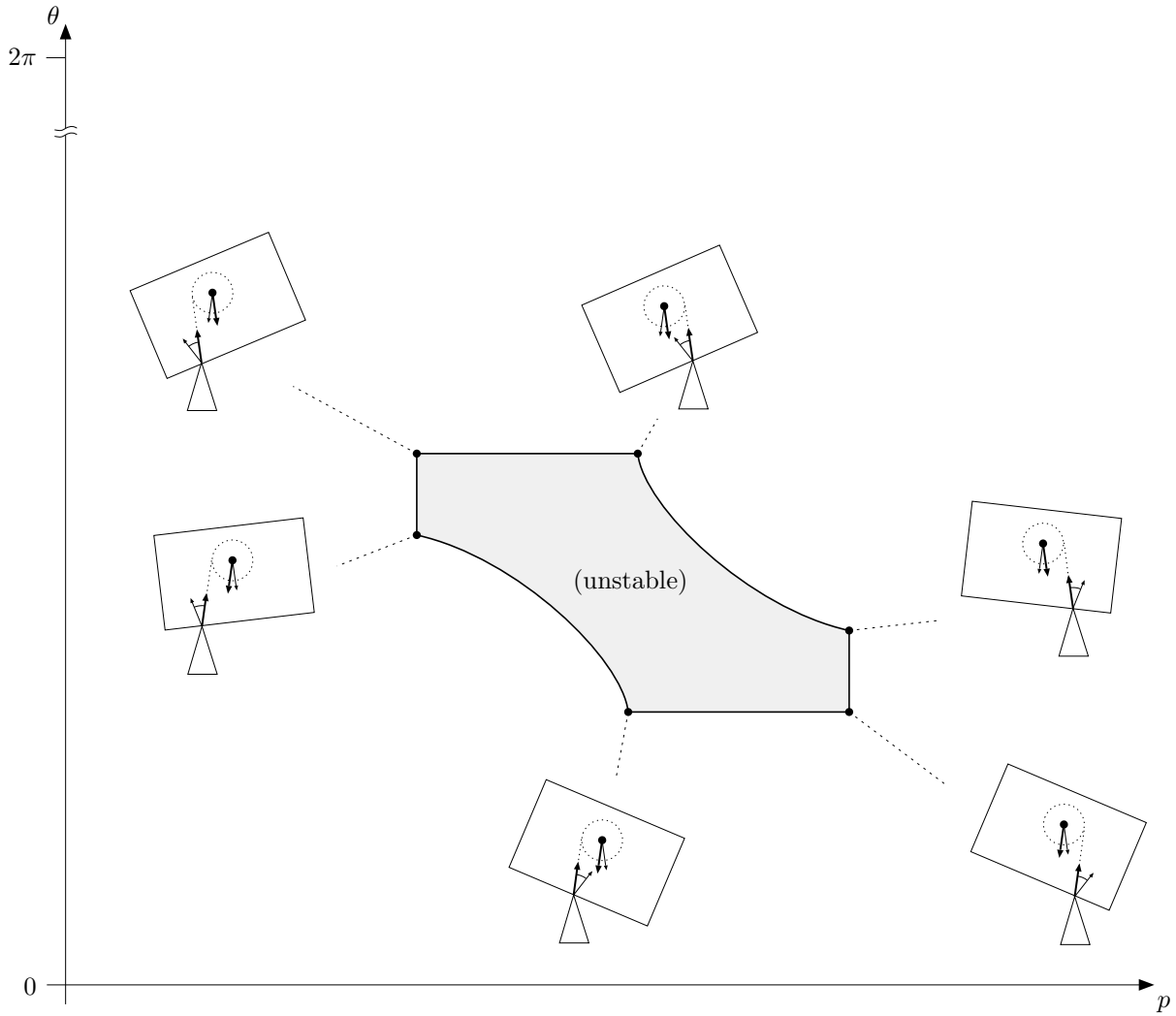


Figure 118: Identifying the possible-equilibrium configurations for an ev facet. The possible-equilibrium subset of the facet c-surface is a bounded region. This region always exists on the ev c-surface, but may be reduced to a single point if there is zero friction and no uncertainty in the applied force. The region may have up to ten bounding edges; only six appear in this example. Intersecting the region with the facet gives the set of reachable possible-equilibrium configurations; in this example, the entire region is reachable. This example region contains only unstable configurations; regions are also possible that contain stable or stable/unstable configurations. See Appendix 4 for a table of equations describing the edges bounding these regions.

Critique

The *STATIC* algorithm constructs the set of all configurations where static equilibrium is possible between two arbitrary polygonal objects, under a specified applied force. The algorithm allows uncertainty in a variety of physical aspects of the system, including:

- Bounded variation in the coefficient of friction μ.
- Bounded variation in the force applied when the system is in static equilibrium.
- Unlimited variation in the force applied when the system is not in static equilibrium.
- Unlimited variation in the mass properties of the objects. Unlimited variation is always allowed in the radius of gyration ρ, and is also allowed in the center of mass whenever the static-equilibrium force is independent of the center of mass (as in compliant motion control, for example).
- Unlimited variation in the microscopic interaction occuring at each vertex-vertex contact, subject to the special assumptions described above. These assumptions require that the microscopic shape of each object vertex is monotonically convex or concave, and that stabilizing microscopic displacements will occur for certain concave vertex-vertex contacts.

This spectrum of uncertainty conditions allows the *STATIC* algorithm to be applied to problems where a complete and precise physical model is not available. However, the algorithm does assume perfect knowledge of the object shapes, and that the objects are perfectly rigid; these strong assumptions limit the algorithm's suitability for some problems.

The results produced by the algorithm are exact, up to the precision of the conservative approximation in the applied-force polygon. This approximation may cause the algorithm to return additional configurations where static equilibrium is not possible, such as the configuration shown in Figure 119. The algorithm will only produce these additional configurations on obstacle edges and vertices, since obstacle facets are analyzed without the conservative approximation.

Removing this approximation would not be difficult; this would require extending the convex-G-polygon representation to include latitude lines, and derivation of additional critical-value equations. However, this modification of the algorithm is of questionable value, for the following reasons: First, the effect of the approximation is small, especially when the error in the applied force is not large. This is emphasized by Figure 110, where a large force error was chosen to make the effect of the approximation clearly visible. Second, the approximation represents a small change to a representation that is already a conservative approximation of the true physical system. For example, a more detailed uncertainty model might constrain the possible applied-force points more precisely than the simple (P_η, r_η) model currently used, or include coupling between errors in the applied-force direction and application point. Such refinements of the applied-force model could continue *ad infinitum,* but would not dramatically reduce the set of possible-equilibrium states returned by the *STATIC* algorithm, except in cases where the applied-force error is very large.

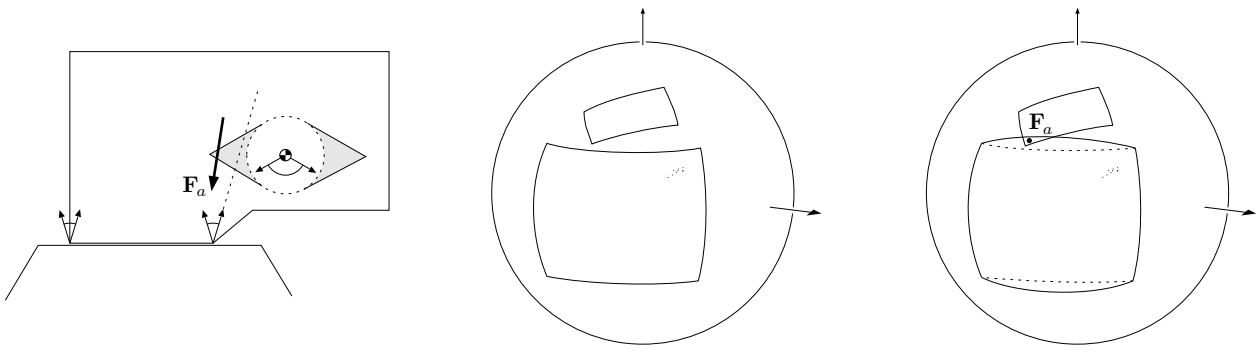


Figure 119: (a) A configuration that would be returned by the *STATIC* algorithm, even though static equilibrium is impossible. This additional configuration is returned as an artifact of the conservative approximation illustrated in Figure 110. (b) The negated configuration-space friction cone and true applied-force polygon. Static equilibrium is impossible because these sets are disjoint. (c) The negated configuration-space friction cone and approximate applied-force polygon. Since these sets intersect, the *STATIC* algorithm concludes that equilibrium is possible. The point $\mathbf{F}_a$ corresponds to the force illustrated in (a).

Implementation and Robustness Issues

I implemented the *STATIC* algorithm and performed experiments to measure its robustness and efficiency. The resulting program is comprised of about 6700 lines of Common Lisp code, excluding debugging and rendering routines, and the code required by the *CO* program.

Like the *CO* program, the *STATIC* program uses floating-point arithmetic, and thus is an approximation of the ideal *STATIC* algorithm. In order to improve the robustness and accuracy of the program, I included several approaches to reducing the impact of numerical calculation errors:

- *Fuzzy arithmetic.* All numeric decision procedures in the *STATIC* program use the same fuzzy-arithmetic operators employed by the *CO* program, with the same numerical tolerance values.
- *Vector normalization.* All $(F_x, F_y, \tau/\rho)$ vectors constructed by the program are normalized as soon as they are constructed. This uniform normalization is required so that point-on-great-circle tests will return consistent results when fuzzy arithmetic is used.
- *Moderate ρ values.* Since static equilibrium is a purely geometric condition, it is independent of the mass properties of the moving-object. Thus ρ can be chosen to be any value, without changing the results of the ideal *STATIC* algorithm. However, the choice of ρ does affect the robustness of the *STATIC* program; extreme values of ρ cause the features on the force sphere to cluster near the poles or equator of the force sphere, leading to numerically-sensitive computations. Thus the *STATIC* algorithm uses a value of ρ roughly near the moving-object's physical radius of gyration, assuming a uniform mass distribution. Since this value is related to the size of the moving-object, it provides a moderate scaling factor for a broad class of objects. Degenerate situations can still occur (for example, if an object edge passes very near the point P_η); these are treated as special geometric cases.

- *Overlapping subcones.* When a contact friction cone is split into two subcones because convex-combinations should not be applied, the subcones are chosen to overlap slightly. This overlap reduces the numerical sensitivity of the corresponding configuration-space friction cones, without compromising correctness.

The resulting program appears to be very robust. However, checking the correctness of the *STATIC* program is more difficult than checking the correctness of the *CO* program, because the set constructed by the *STATIC* algorithm does not have the strong topological structure that characterizes the configuration-obstacle surface. Metric tests are also difficult, because of the nature of the constructed set: For every point returned by the algorithm, there must exist some choice of uncertainty conditions and force-magnitudes where static equilibrium occurs; similarly, for every point excluded from the algorithm, static equilibrium should never be possible, for any choice of uncertainty conditions or force magnitudes. Verifying these conditions is a difficult computational problem, so I did not implement an automatic test procedure for the *STATIC* program. Instead, I manually checked the algorithm using a collection of interactive test programs, paying special attention to the various boundary conditions that can arise. These manual test were further reinforced by the physical experiments described in Part I.

As with the *CO* program, several optimizations were included to improve the performance of the *STATIC* program. These optimizations are similar to those used in the *CO* program, and include incremental computation and simple decision procedures to determine when expensive analysis steps can be trivially avoided. I measured the performance of the algorithm over a suite of 30 test cases, most of which addressed the *STATIC* program's ability to handle various boundary conditions correctly. The following table shows execution times for some of the examples that appear in this thesis:

Example	*CO* (seconds)	*STATIC* (seconds)
Figure 8 (4 facets)	3.7	8.6
Figure 20 (Full obstacle)	124.5	56.5
Figure 111 (Full obstacle)	8.6	30.6
Figure 18 (8 facets)	4.5	7.1
Figure 19 (3 facets)	2.4	6.1

The last three entries of this table correspond to the same pair of input objects, analyzed under different conditions. These examples reflect typical execution times for the analysis of full and partial obstacles; the total execution time is the sum of the *CO* and *STATIC* execution times. The execution time tended to grow as the friction and uncertainty increased; this was primarily due to the fact that fewer computations could be trivially avoided.

One lesson learned from the implementation is that explicit representations of configuration-space sets are far more useful than implicit representations. The data structure employed by the *CO* algorithm is an example of an explicit representation, since all obstacle features and their extent are explicitly represented. In contrast, the set returned by the *STATIC* program is implicit, where each set of reachable possible-equilibrium configurations is represented as an intersection of an obstacle feature and a set of possible-equilibrium features. The significance of this becomes apparent for ev facets, where the reachable and possible-equilibrium configurations may be disjoint, but this is not detected. This proved to be a handicap for later applications of the algorithm, so modifying the *STATIC* program to return an explicit representation would be an improvement.

Extending the Algorithm to Perform Reliable-Equilibrium Analysis

The *STATIC* algorithm constructs the set of configurations where equilibrium is possible; can we also construct the set of configurations where equilibrium reliably occurs? Perhaps, but first we must define the notion of "reliable equilibrium." Here are some possibilities:

1. When the system is in a given configuration, it will remain at rest in the configuration.
2. When the system is at rest in a given configuration, it will remain at rest in the configuration.
3. When the system is at rest in a given configuration, it will remain at rest in the configuration, even in the presence of small perturbations.

Definition 1 is the simplest, but constructing the set of reliable-equilbrium states under this definition requires information describing the system's mechanical behavior, both at rest and in motion. This is required to assure that when the system is in a reliable-equilibrium configuration and initially in motion, it will immediately come to rest. However, no configuration will ever satisfy this criterion, unless the system is quasi-static or perfectly plastic. This is because the configuration might have been reached as an intermediate state in a high-velocity collision; such a state will generally be followed by an immediate rebound motion.

Definition 2 is more reasonable, since it does not require a model of the system's behavior when in motion, and also does not admit the possibility of collisions. However, it will predict that unstable equilibrium configurations will reliably remain at rest, which is implausible for most physical systems (Figure 120).

Definition 3 includes stability as a criterion for reliable equilibrium, and thus produces a more plausible prediction for the scenarios illustrated in Figure 120. However, things become muddy again when uncertainty is included, as shown in Figure 121. In this scenario, stable equilibrium will reliably occur somewhere in the shaded interval, but the exact configuration cannot be determined. What should the reliable version of the *STATIC* algorithm return? One possibility might be to return the entire interval of orientations, with an associated flag indicating that equilibrium will reliably occur at some unknown orientation within the interval. But then what if only part of this interval is reachable, as shown in Figure 121(c)?

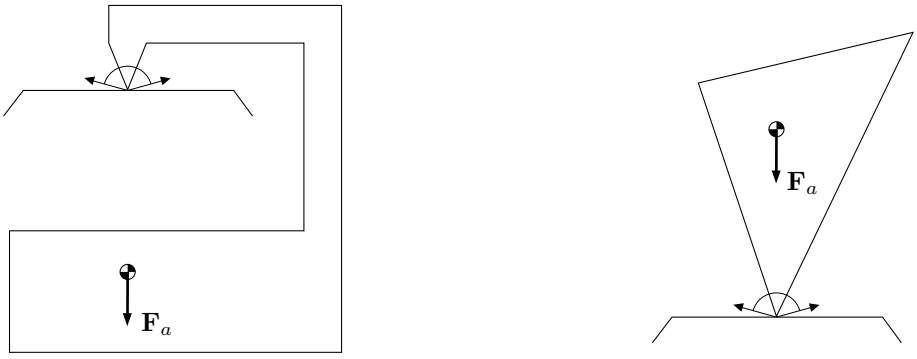


Figure 120: Stable and unstable static equilibrium situations.

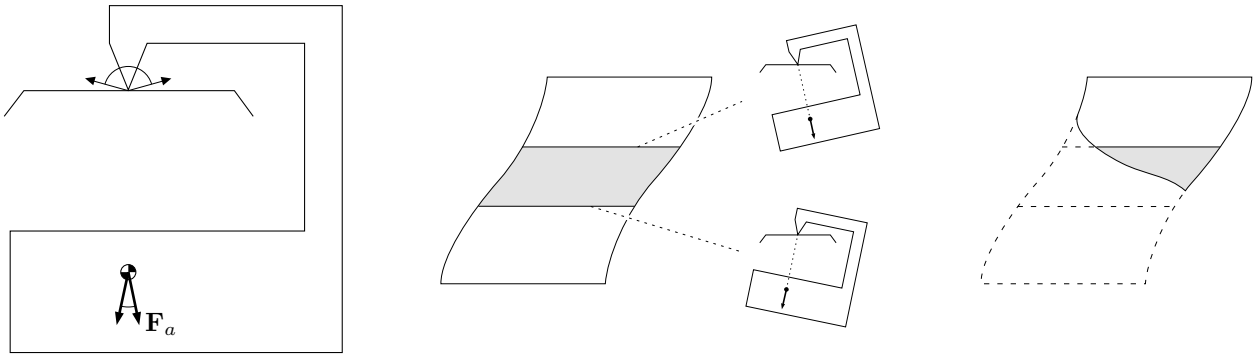


Figure 121: A stable static equilibrium situation, with uncertainty. (a) The contact condition. (b) The corresponding facet in the configuration space. Static equilibrium will reliably occur somewhere in the shaded region, but the exact configuration is unknown because it depends on the uncertain applied force direction. (c) A similar situation, but where part of the reliable-equilibrium region is unreachable.

These considerations reveal that construction of reliable-equilibrium configurations is a semantically slippery issue. Preferences among these and other possible definitions of reliable equilibrium will affect the algorithm used to construct these sets; these preferences will be closely tied to the particular physical system being analyzed.

Once these aspects have been determined, it may be possible to extend the *STATIC* algorithm to construct the desired reliable-equilibrium states. For example, it would be straightforward to modify the *STATIC* algorithm to construct the set of configurations $\mathcal{C}_{\text{contain}}$ where the applied-force polygon is fully contained within the reliable configuration-space friction cone. As mentioned above, this is a necessary but not sufficient condition for static equilibrium to reliably occur; therefore the set of reliable-equilibrium configurations must be a subset of $\mathcal{C}_{\text{contain}}$. A dynamic analysis could then be applied to discard ambiguous configurations where motion is also possible; the techniques presented in [Erdmann 1984] or [Brost and Mason 1990] provide possible approaches.

Another approach would be to adopt the view that the reliable-equilibrium states of interest are those where the system will come to rest from a given set of initial conditions. This view relegates the reliable-equilibrium question to the purvey of dynamic analysis, and identifies a set of reliable-equilibrium states that are utilitarian in the sense that equilibrium not only occurs reliably, but can be achieved reliably. This is the point of view adopted in this thesis, and the necessary dynamic analysis is discussed in the next chapter.

Dynamics

Classical dynamics addresses the following questions:

- Given a physical system, an applied force, and an initial state, what is the time-history of subsequent states?
- Given a physical system, an applied force, and a final state, what is the time-history of preceding states?

We may choose to ignore the time-history information associated with these questions, and only consider the set of states visited by the system before or after a specified state. This amounts to considering the path of the system in state space, rather than the trajectory of the system. This simplification modifies the above questions:

- Given a physical system, an applied force, and an initial state, what is the set of states that will follow the initial state?
- Given a physical system, an applied force, and a final state, what is the set of states that will precede the final state?

All of the above questions are relevant to the analysis and planning of manipulation tasks, because they correspond to predicting the outcome of a manipulation action, and synthesizing manipulation actions to achieve a desired goal. However, these questions do not capture the uncertainty that is inherently present in manipulation tasks. We can re-frame these questions to include uncertainty as follows:

- Given a physical system, a set of possible applied forces, and a set of possible initial states, what is the set of possible subsequent states?
- Given a physical system, a set of possible applied forces, and a set of desired final states, what is the set of preceding states that will reliably produce one of the desired final states?

These questions define the *forward projection* of a set of initial states, and the *strong backprojection* of a set of desired final states; these notions were previously explored in [Lozano-Pérez *et al.* 1984] and [Erdmann and Mason 1986]. This chapter will focus primarily on the strong backprojection, formulated through a modified version of the above question:

- Given a physical system, an applied action, a set of desired goal states $\mathcal{G}$, and a set of constraint states $\mathcal{C}$, what is a set of preceding states V_{state} that will reliably produce a final state in $\mathcal{G}$, never encountering a state in $\mathcal{C}$?

This question differs from the preceding question in several key respects. First, the notion of an applied force is replaced by an applied action, which is an abstraction that simplifies the specification of the physical system and its dynamical behavior. Second, the set $\mathcal{C}$ augments the previous specification of desired system behavior by constraining the set of acceptable paths to the final state. Finally, this question is less restrictive than the previous question, since it asks for "a set" rather than "the set" of satisfactory preceding states. This allows subsets of the maximal strong backprojection as acceptable solutions.

The BP_e and BP_i algorithms defined in Figure 12 of Part I address the problem of constructing strong backprojections under this definition. These algorithms are specifically concerned with physical systems that are comprised of two polygonal objects, where the (x, y, θ) relative configuration of the objects provides sufficient state information for dynamic analysis. As of this writing, these algorithms have not been fully developed or implemented. In the sections that follow, we will present the geometric constructions that the algorithms employ to identify strong backprojections, and sketch procedures for implementing these constructions. Some of the details are missing from these sketches; these are left for future work.

Constructing V_{state} Using Energy Arguments

The BP_e algorithm constructs strong backprojections for physical systems whose behavior can be bounded by an invariant analogous to the conservation of mechanical energy. These systems include objects falling in a gravity field, and objects controlled by certain compliant-motion control laws, such as generalized damper control.

To gain an intuitive understanding of the invariant, consider the simple example of a particle falling in a gravitational field amidst polygonal obstacles (Figure 122). In this physical system, the energy of the particle is bounded by the potential energy of its release height. Once the particle is released from a height y_o, it can never reach a height greater than y_o. This implies that the particle can follow a complex trajectory involving thousands of collisions, but the particle will never leave the half-space defined by $y < y_o$.

We can view the bounded energy of the particle as a geometric constraint defined at the particle's release position. By placing the particle in the initial position and constructing a line perpendicular to the gravity direction, we can identify a half-space corresponding to the set of all reachable particle positions (Figure 123). This construction gives us a geometric interpretation of the potential-energy bound, expressed in the configuration space of the particle.

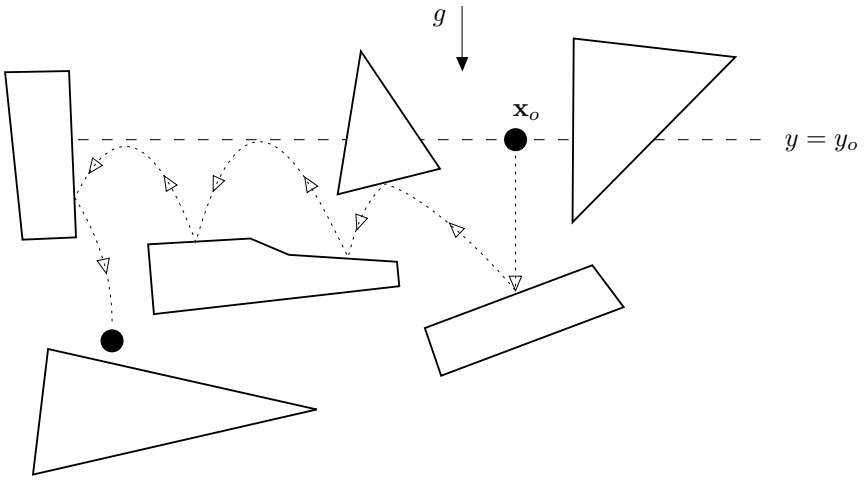


Figure 122: Dropping a particle amidst fixed obstacles. The particle may experience numerous collisions, resulting in a complex trajectory. However, the trajectory will never pass above the $y = y_o$ line (shown dotted), because the particle's mechanical energy is bounded.

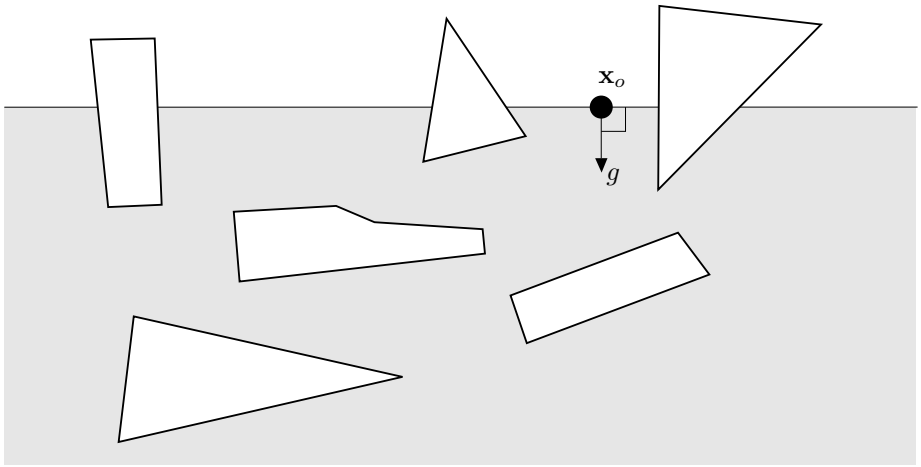


Figure 123: Expressing the mechanical energy bound as a geometric constraint in the configuration space. The particle might reach any point in the shaded region, but will never reach a point outside the region.

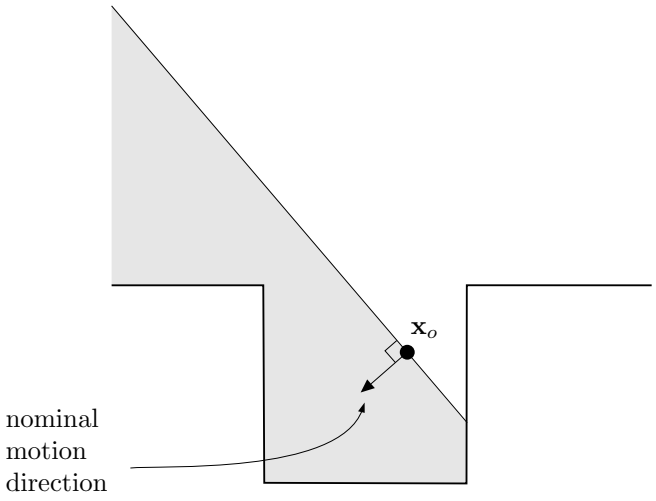


Figure 124: A "potential energy" constraint for generalized damper control. (a) This motion control law will never backtrack relative to its nominal motion direction, so it can never escape the half-space defined by the normal to this direction and the initial position $\mathbf{x}_o$. This constraint is weaker than the forward-projection described in [Lozano-Pérez *et al.* 1984], which utilizes a more detailed model of the task mechanics.

Generalizing the Notion of Mechanical Energy

We can generalize this construction to apply to other systems where a similar constraint can be identified. Compliant motion under generalized damper control is one example (Figure 124). Uncertainty in the physical system may also be included, as shown in Figure 125.

These observations lead us to define the following invariant that must be satisfied to support the half-space constructions described above:

> *The Half-Space Invariant:* If the system is initially at rest in a configuration $\mathbf{x}_o$, then all subsequent configurations must lie in the half-space defined by a configuration-space hyperplane normal to η_{true} and containing P_{true}, where η_{true} is a direction in $[\eta_{\min}, \eta_{\max}]$, and P_{true} is a point within a distance r_η of $\mathbf{x}_o$.

This invariant is defined in generic terms, and may be shown to apply to a variety of physical systems. For example, the invariant holds for a rigid body falling in a gravity field amidst fixed obstacles, if the origin of the body is chosen to be within r_η of the center of mass.

This invariant makes very weak assumptions about the system's dynamic behavior; arbitrary trajectories are allowed, as long as the trajectory is confined to the appropriate half-space in the configuration space. This is at once a dramatically powerful and limiting assumption. Because the requirement is so weak, we can analyze some very complex physical systems, such as systems involving dynamic collisions. However, this weak mechanics model also prevents the BP_e algorithm from finding solutions to apparently simple problems, as we shall see later.

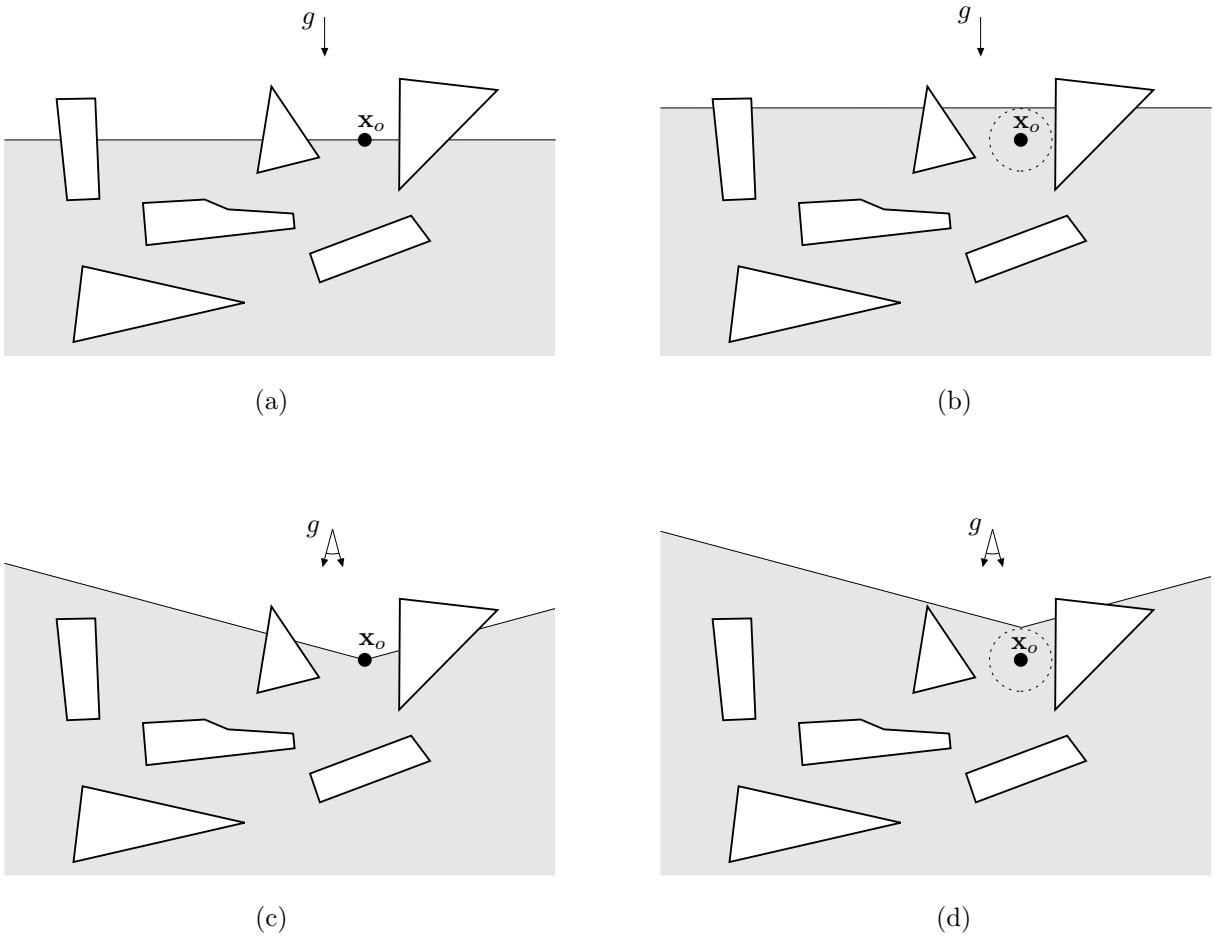


Figure 125: The effect of uncertainty on the set of reachable configurations. (a) The set of reachable configurations after releasing the particle from an initial configuration $\mathbf{x}_o$, without uncertainty. (b) If the particle's center of mass is uncertain, the half-space must be expanded to include the highest possible initial center-of-mass position. (c) If the gravity direction is uncertain, the half-space must be expanded to include all possible orientations of the half-space constraint. (d) The combined effect of both sources of uncertainty; if the particle is initially released from $\mathbf{x}_o$, it cannot escape the shaded region, regardless of the choice of uncertainty conditions.

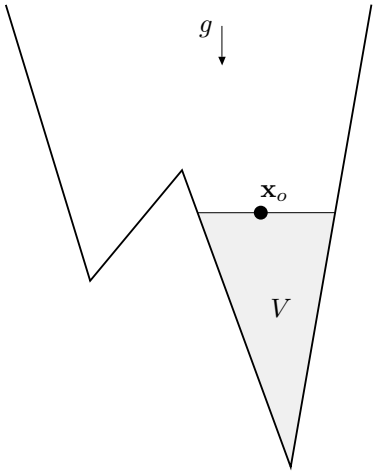


Figure 126: Combining half-space and kinematic constraints. After release from an initial position $\mathbf{x}_o$, the particle is constrained to the shaded region V.

The Pigeonhole Principle

The half-space constraints developed above may combined with kinematic constraints to further localize the set of reachable configurations for a physical system satisfying the half-space invariant. Figure 126 illustrates this for our falling-particle example. Here the particle is constrained to the shaded region V, since it cannot reach other portions of the half-space without gaining mechanical energy.

This result may be combined with a possible-equilibrium analysis to identify the final position of the particle. The particle can only come to rest in a configuration where static equilibrium is possible; otherwise a force imbalance would occur. If V contains only one such configuration, then the final position of the particle is uniquely determined (Figure 127).

Further, the particle must come to rest in a finite amount of time. This assertion is a direct consequence of the third law of thermodynamics, which implies that perfectly elastic materials and frictionless contacts cannot exist in real physical systems. These observations imply the following:

> Let W be the intersection of the half-space constraints and the kinematically reachable configurations of a physical system satisfying the half-space invariant. Let $V \subseteq W$ be the connected component of W containing the initial configuration. If V is bounded, then the system must come to rest in one of the possible-equilibrium configurations within V.

Therefore we can predict the final state of a physical system given its initial state, even though we know virtually nothing about the trajectory that connects the initial and final states.

This observation forms the foundation of the BP_e algorithm: Given a goal $\mathcal{G}$ and an action that satisfies the half-space invariant, the algorithm attempts to find the highest half-space constraint in configuration space that encloses a connected region V where all possible-equilibrium configurations within V are goal configurations. V will correspond to a strong backprojection of $\mathcal{G}$, since any initial configuration within V will come to rest in a configuration in $\mathcal{G}$.

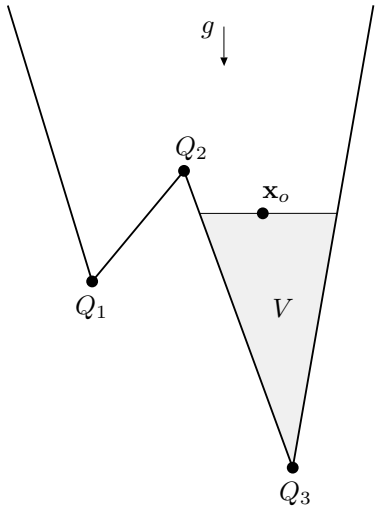


Figure 127: Q_1, Q_2, and Q_3 show the possible-equilibrium configurations for the particle, while V shows the set of reachable configurations for the particle after initial release from the position $\mathbf{x}_o$. The particle must come to rest in configuration Q_3.

Representing "Puddles" in Configuration Space

The backprojections formed by the BP_e algorithms are volumes in configuration space that are analogous to "puddles" of water on the configuration obstacle surface. These puddles are connected sets of configuration-space points that are kinematically reachable and also satisfy one or more half-space constraints.

The boundary of each puddle is comprised of two distinct surfaces: The "top" and "bottom" of the puddle. The top of the puddle corresponds to the flat surface formed by the half-space constraint(s), while the bottom of the puddle corresponds to the surface formed by the interface between the puddle and the configuration-space obstacle. We can represent the entire puddle surface by representing these two surfaces and their adjacency relationship.

The bottom of the puddle is easy to represent. Since each portion of this surface corresponds directly to a configuration-obstacle feature, we can use the normal c-obstacle data structure, applying the necessary transformations to turn the obstacle surface inside out.

The top of the puddle is also straightforward. The planar half-space constraints in the (x, y, θ) configuration space correspond to degenerate ve facets; these facets arise on the surface of a configuration obstacle whenever a vertex of the moving-polygon is coincident with the moving-polygon origin. We can construct these facets by constructing an imaginary fixed-edge at the desired position and orientation, and an imaginary vertex at the moving-polygon origin. These imaginary features define an infinite c-surface corresponding to the planar half-space constraint. Further, the intersection of the half-space constraint with the configuration obstacle is comprised of ve-ev and ve-ve obstacle edges that have already been derived (see Appendix 3).

Thus we can represent a configuration-space puddle using a slightly modified version of the c-obstacle data structure developed previously. It is convenient to include additional control fields in this data structure (such as a flag indicating which facets correspond to half-space constraints); we will omit these details here.

The BP_e algorithm uses five operations to construct puddles in configuration space:

- *Initialization.* A measure-zero puddle is constructed to include an input set of locally-minimal configurations; this operation is trivial.
- *Expansion.* Water is added to a puddle until it reaches a specified height. In configuration space this amounts to constructing a ve facet c-surface corresponding to the specified water level, intersecting this facet with the obstacle surface, and selecting the patch from the resulting collection of patches that is part of a connected surface containing the initial puddle.
- *Merging.* Adjacent puddles are combined before additional expansions are performed; this is a purely topological operation.
- *Intersection.* Two puddles are intersected to include uncertainty in the applied-force direction. This intersection is much simpler than a general configuration obstacle intersection, since the bottom surfaces of each puddle correspond to identical obstacle features, and the top surfaces are planar ve facets.
- *Retraction.* This is similar to puddle expansion, except water is removed from a puddle instead of added. Special code must be included to properly handle cases where the puddle is completely drained.

These operations are similar to the constructions previously described for the *CO* and *STATIC* algorithms. Because of the similarity between configuration-space puddles and configuration-space obstacles, implementing these construction operations should be a relatively straightforward extension of the construction procedures employed in the *CO* algorithm. Partial implementations have been developed as of this writing; the remaining details are left for future work.

The BP_e Algorithm

The BP_e algorithm accepts the following input:

- A polygonal moving-object $\mathcal{P}_m$.
- A polygonal fixed-object $\mathcal{P}_f$.
- A table of friction coefficients $\mathcal{T}_\mu$.
- A set of possible applied forces $\mathcal{F}_a$, described by an interval of force directions $[\eta_{\min}, \eta_{\max}]$, and a nominal force application point P_η, and an error distance r_η. To simplify the following discussion, we will assume that P_η is coincident with the origin of $\mathcal{P}_m$.
- A set of desired goal configurations $\mathcal{G}$.
- A set of task constraints $\mathcal{C}$, defining a set of configurations that should never occur.

Given this input, the algorithm constructs a set of configurations V_{state} that will reliably achieve a final configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{C}$. The BP_e algorithm constructs V_{state} by identifying the largest puddle on the configuration-space obstacle that contains reliably-stable configurations in $\mathcal{G}$, but no undesirable configurations (that is, possible-equilibrium configurations not in $\mathcal{G}$, or configurations in $\mathcal{C}$). Figure 128 shows several example results produced by the algorithm, and Figures 129 through 131 summarize the algorithm in pseudo-code.

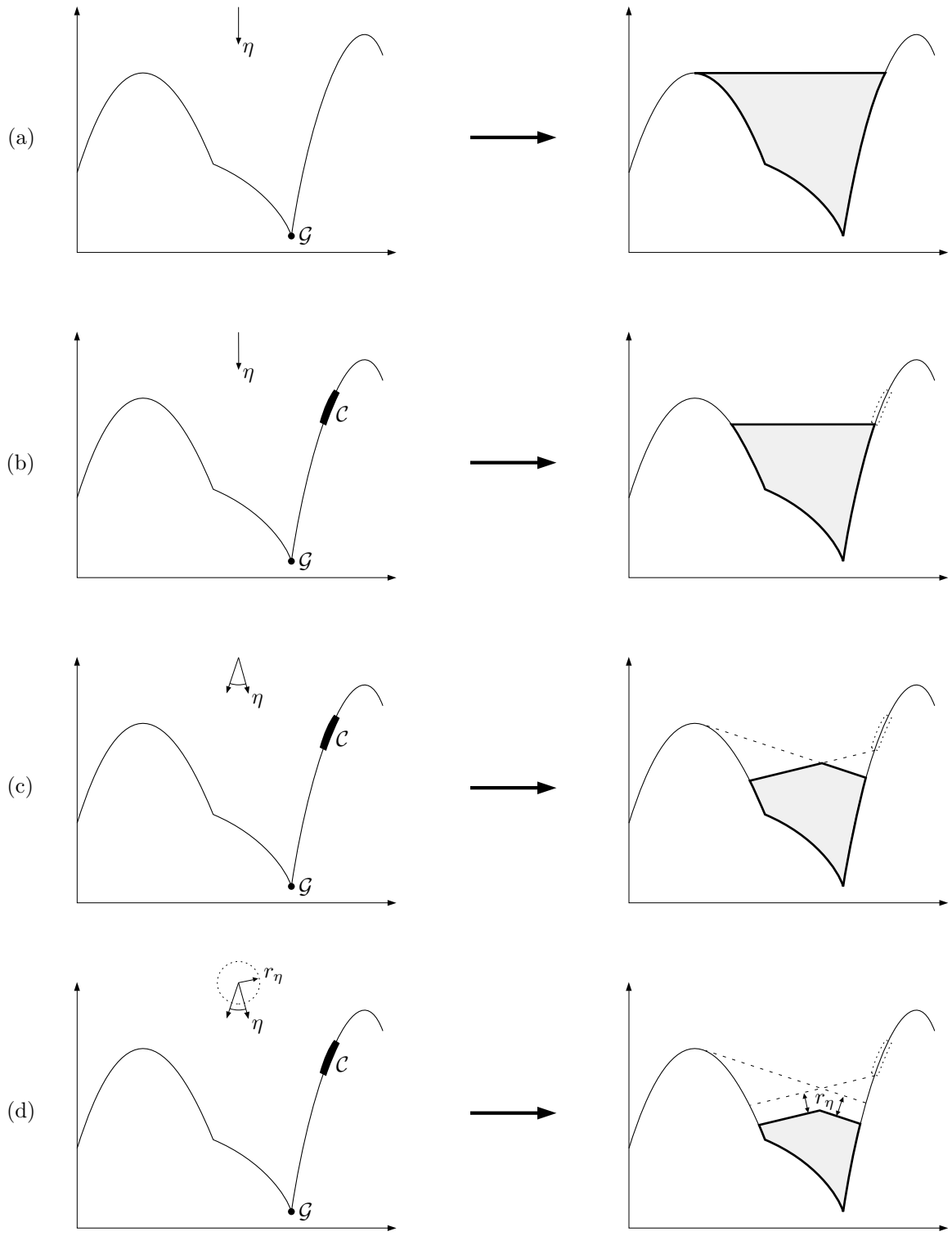


Figure 128: Example backprojections constructed by the BP_e algorithm. The coordinate axes show a generic configuration space. The diagrams on the left show the input, and the diagrams on the right show the algorithm's output. (a) A simple case. (b) Including a constraint set $\mathcal{C}$. (c) Adding uncertainty in the applied force direction. (d) Adding uncertainty in the applied force direction and application point.

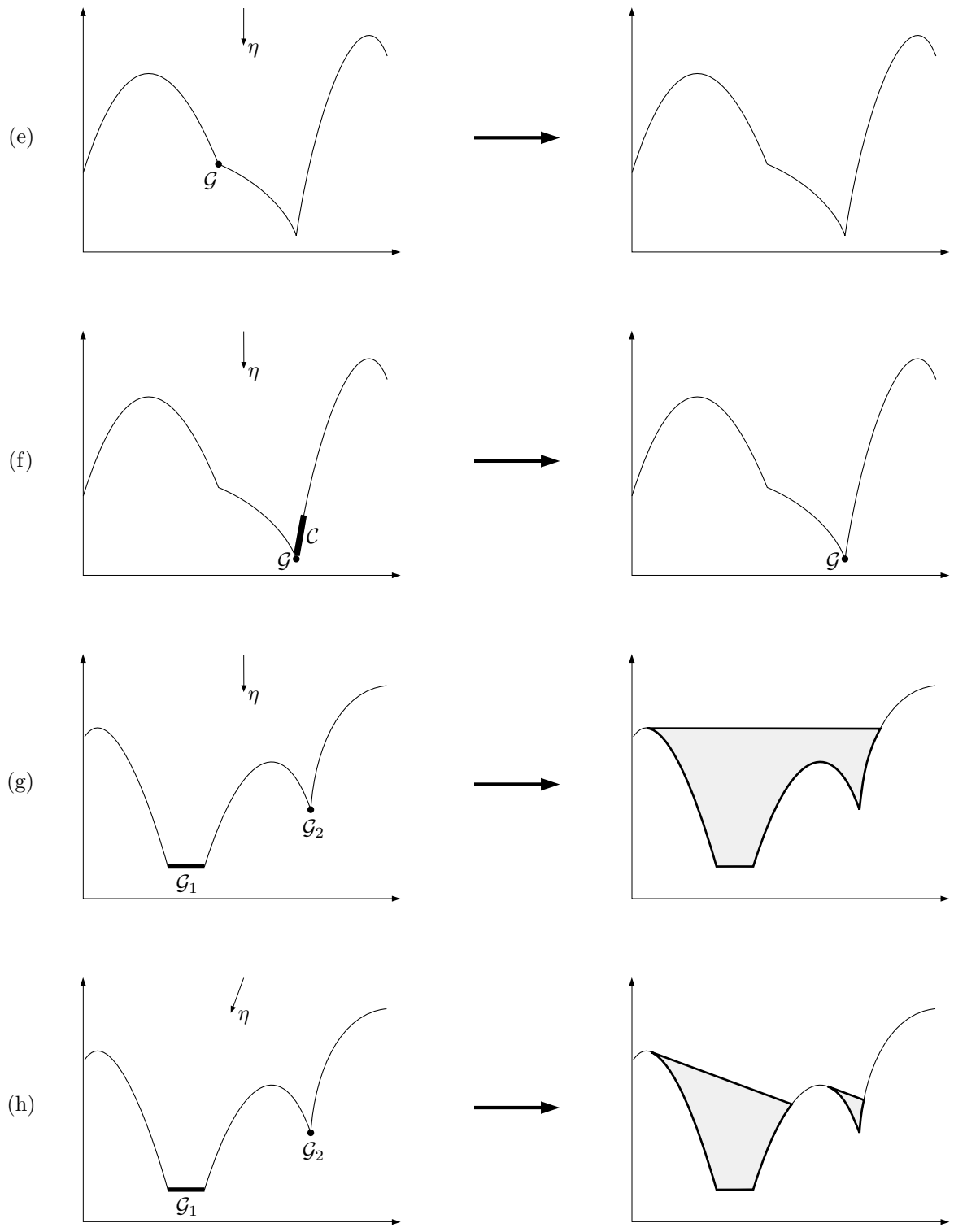


Figure 128 (continued): (e) The unstable goal $\mathcal{G}$ produces a null backprojection. (f) The stable goal $\mathcal{G}$ is adjacent to the constraint set $\mathcal{C}$; the strong backprojection is only the point $\mathcal{G}$. (g) A goal comprised of two disjoint sets $\mathcal{G}_1$ and $\mathcal{G}_2$, yielding a large connected backprojection. (h) The same example as in (g), but with a different force direction; the strong backprojection is now disconnected.

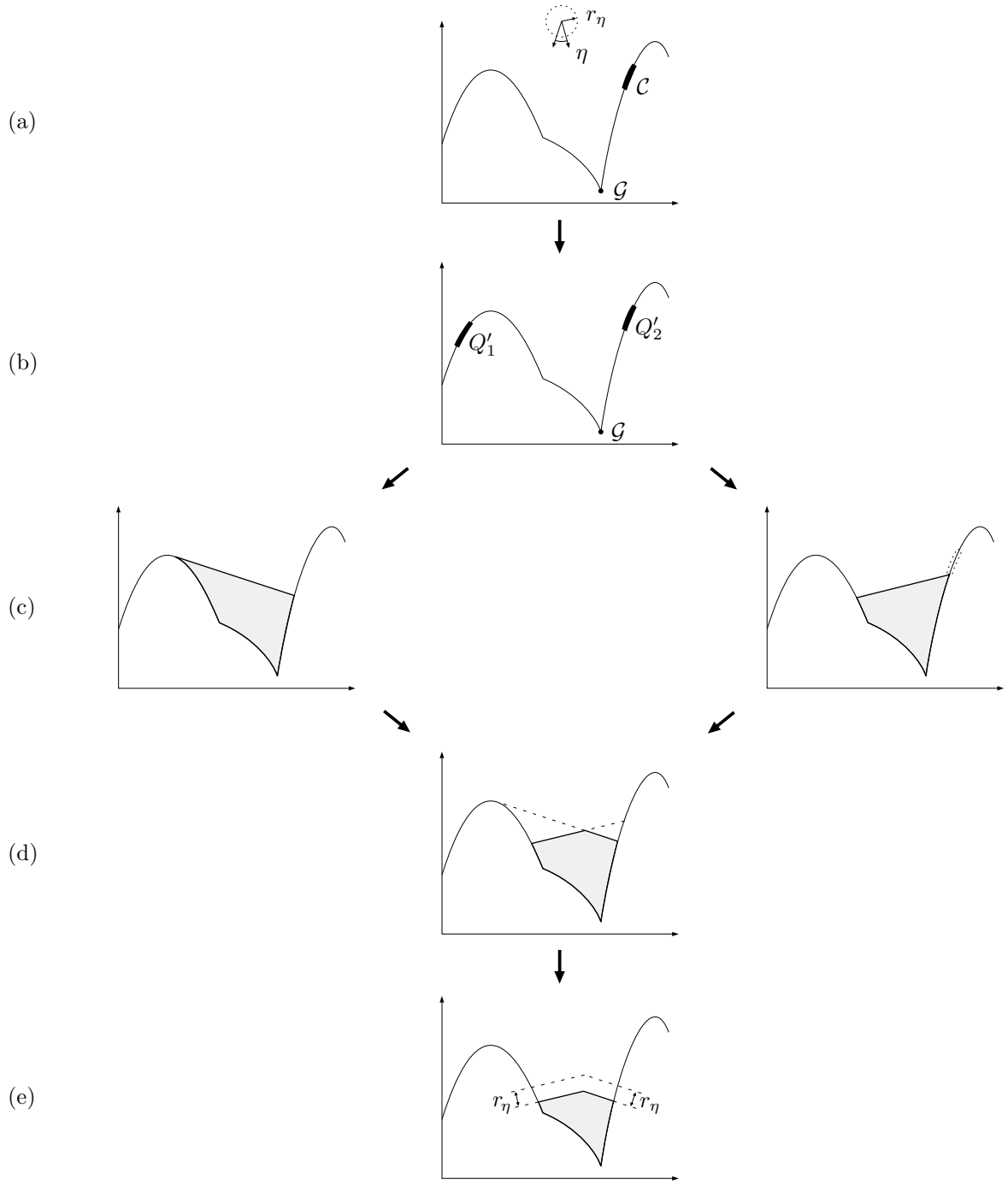


Figure 129: Constructing the strong backprojection of a goal, with uncertainty. (a) The input. The configuration-space obstacle of the two input objects is shown, along with the applied-force uncertainty. $\mathcal{G}$ is the goal configuration, and $\mathcal{C}$ is a set of constraint configurations that should be avoided. (b) After constructing $\mathcal{Q}'$, the abstracted set of configurations that should be avoided. Q_1' is a set of possible-equilibrium configurations, while Q_2' is the constraint set $\mathcal{C}$. (c) After constructing the strong backprojection for $\eta_{\min}$ and $\eta_{\max}$. (d) After intersecting the $\eta_{\min}$ and $\eta_{\max}$ backprojections, producing a strong backprojection for all directions in $[\eta_{\min}, \eta_{\max}]$. (e) After shrinking to account for r_η, the uncertainty in the applied-force point.

$\mathbf{BP}_e(\mathcal{P}_m, \mathcal{P}_f, \mathcal{T}_\mu, \mathcal{F}_a, \mathcal{G}, \mathcal{C})$

Construct the configuration obstacle.
Construct the configuration-space obstacle $\mathcal{O}$ of the polygonal objects $\mathcal{P}_m$ and $\mathcal{P}_f$, using the *CO* algoirthm. Assure that $\mathcal{G}$ and $\mathcal{C}$ are disjoint by removing configurations from $\mathcal{G}$ that are also in $\mathcal{C}$.

Construct the possible-equilibrium states.
Construct the set of configurations $\mathcal{Q}$ where static equilibrium is possible, using the *STATIC* algorithm.

Construct $\mathcal{Q}'$.
Construct the set of avoidance configurations $\mathcal{Q}' = (\mathcal{Q} - \mathcal{G}) \cup \mathcal{C}$. This is the set of configurations that should not be included in any strong backprojection.

Construct strong backprojection, including direction uncertainty.
If $\eta_{\min} = \eta_{\max}$, then

$$V \leftarrow BP_{e_{\text{ideal}}}(\mathcal{O}, \eta_{\min}, \mathcal{G}, \mathcal{Q}')$$

else

$$V_{\min} \leftarrow BP_{e_{\text{ideal}}}(\mathcal{O}, \eta_{\min}, \mathcal{G}, \mathcal{Q}')$$
$$V_{\max} \leftarrow BP_{e_{\text{ideal}}}(\mathcal{O}, \eta_{\max}, \mathcal{G}, \mathcal{Q}')$$
$$V \leftarrow V_{\min} \cap V_{\max}$$

Add uncertainty in the applied-force point.
Retract the half-space constraints defining V by a distance r_η, producing a smaller set of configurations V_{state}. This retraction may cause some components of V to vanish; special code must be included to retain goal configurations that are reliably stable but have measure-zero backprojections.

Return V_{state}.
Return the configuration set V_{state}, which is a set of configurations that will reliably achieve a final configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{C}$. V_{state} includes the effects of uncertainty in friction, the direction of the applied force, and the point of application of the applied force.

Figure 129 (continued): The BP_e algorithm.

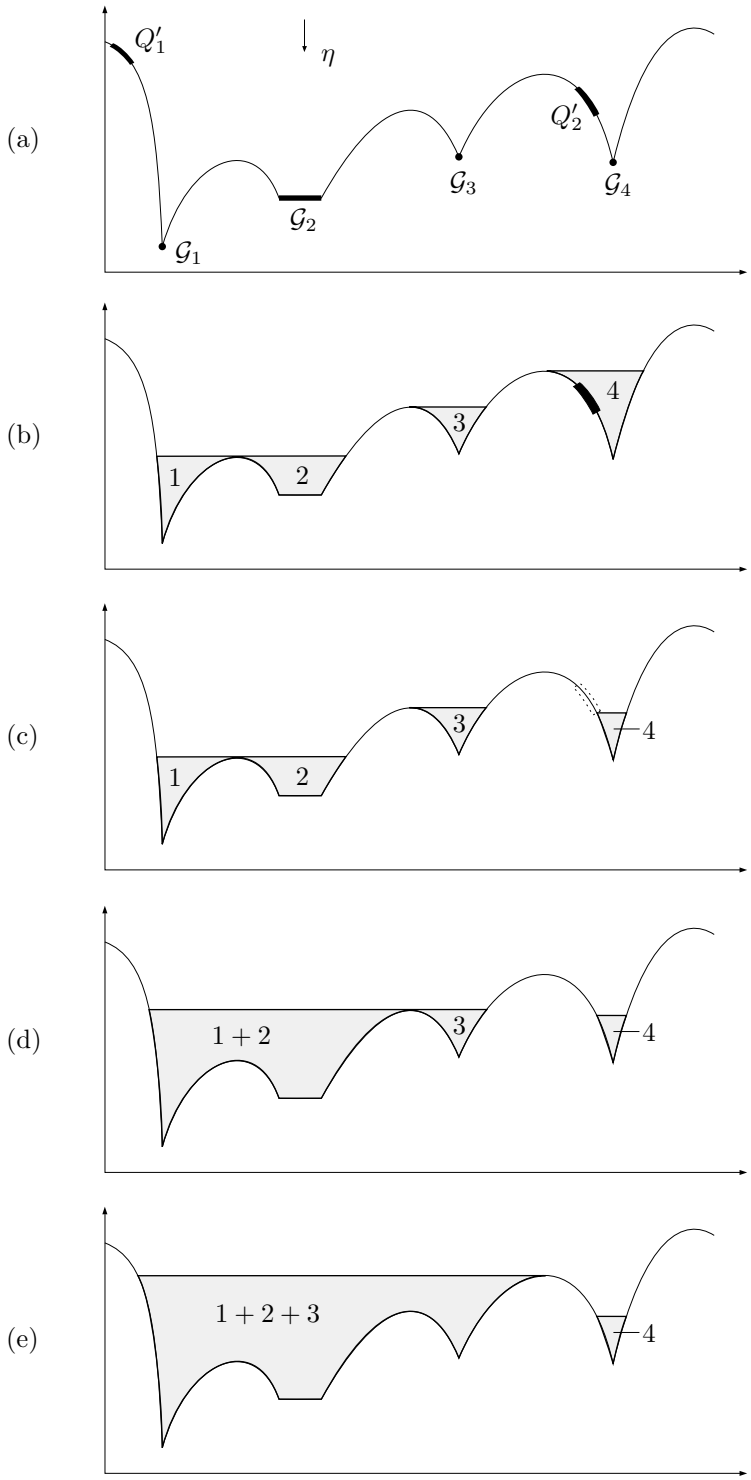


Figure 130: Constructing the backprojection of a goal, without uncertainty. (a) The input; Q'_1 and Q'_2 are undesirable configurations. (b) After the first kinematic expansion step. (c) After shrinking to eliminate contained Q' states. (d) After merging puddles 1 and 2 and repeating the expansion. (e) After merging puddles 1+2 and 3 and repeating the expansion; this is the final result.

$\mathbf{BP}_{e_{\text{ideal}}}(\mathcal{O}, \eta, \mathcal{G}, \mathcal{Q}')$

This procedure constructs a strong backprojection of $\mathcal{G}$, assuming that the applied force is precisely known, and that the constraint configurations $\mathcal{C}$ and non-goal possible-equilibrium configurations are represented by the abstracted set $\mathcal{Q}'$.

Identify local minima.
Using the techniques described in the text, identify the set of points $\mathcal{M}$ on the configuration obstacle $\mathcal{O}$ that correspond to local minima on the surface, with respect to the force-direction η.

Identify goal minima.
Construct the set $\mathcal{G}' = \mathcal{M} \cap \mathcal{G}$ of local-minima goal configurations. If $\mathcal{G}' = \emptyset$, return.

Construct initial puddles $\mathcal{V}$.
For every connected set of points $G' \in \mathcal{G}'$, construct a puddle of zero volume, with an elevation equal to the height of G'.

Expand the set of puddles as far as possible.
Repeat:

> **Merge adjacent puddles in $\mathcal{V}$.**
> For every pair of puddles $(V_1, V_2) \in \mathcal{V}$ that share a common boundary point, merge V_1 and V_2 to form V', updating $\mathcal{V}$ appropriately.
>
> **Expand each puddle in $\mathcal{V}$.**
> For each puddle $V \in \mathcal{V}$:
>
> > **Expand V up to kinematic limits.**
> > Expand the puddle V until water would "spill over" into another component of the configuration space. This is accomplished by calling the procedure *kinematic-BP*$_{e_{\text{ideal}}}(\mathcal{O}, \eta, V)$.
> >
> > **Shrink V to eliminate contained $\mathcal{Q}'$ states.**
> > Form the set $\mathcal{Q}'' = \mathcal{Q}' \cap V$. If $\mathcal{Q}'' \neq \emptyset$, reduce the height of V to $h_{\mathcal{Q}''}$, the minimum height of a point in $\mathcal{Q}''$. This may shrink the puddle back to its previous height; these cases may be detected earlier by noticing that $\mathcal{Q}'$ was adjacent to the boundary of V before expansion.

until no adjacent puddles may be merged in $\mathcal{V}$.

Return the union of the puddles in $\mathcal{V}$.
Under the assumption that the applied force is precisely known, the puddles in $\mathcal{V}$ represent a volume of initial configurations that will reliably achieve a final configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{Q}'$.

Figure 130 (continued): The *BP*$_{e_{\text{ideal}}}$ procedure.

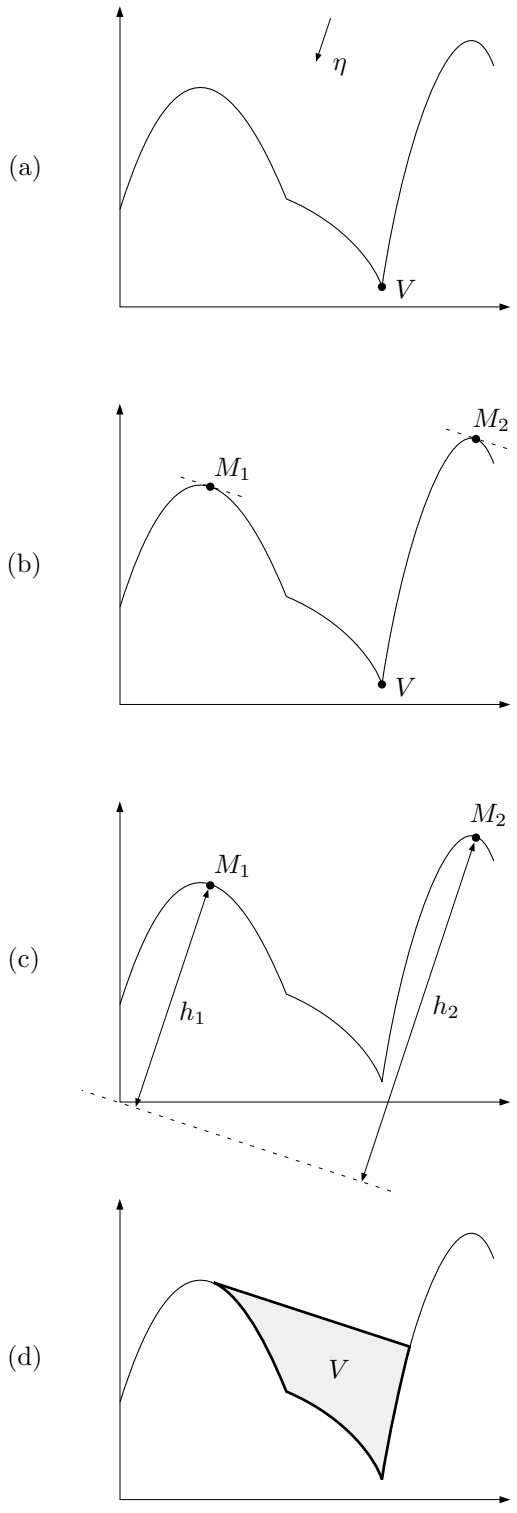


Figure 131: Expanding a puddle V to its kinematic limits. (a) The input: A configuration obstacle, a single direction η, and a puddle V. (b) Relevant spillover points. (c) The elevation of each spillover point, measured relative to the direction η. Since $h_1 < h_2$, M_1 is the limiting spillover point. (d) After expanding V to h_1.

kinematic-BP$_{e_{\text{ideal}}}(\mathcal{O}, \eta, V)$

This procedure expands the puddle V until its water level reaches a critical point on the obstacle surface, such that adding more water would cause the puddle to spill over into another component of the configuration space. These critical points correspond to local maxima of the height function defined by η; the puddle may be expanded until the first such local maximum is encountered.

Identify relevant spillover points.
Using the techniques described in the text, identify the set of points $\mathcal{M}$ that correspond to spillover points that constrain the puddle V. These points correspond to saddle points or local maxima that would cause expansions of V to spill into a new component of configuration space.

Compute elevation of spillover points.
Compute the height value h for each spillover point $(x, y) \in \mathcal{M}$, measured relative to the direction η. This is accomplished by applying the formula:

$$h(x, y) = -\begin{bmatrix} x \\ y \end{bmatrix} \cdot \begin{bmatrix} \cos(\eta) \\ \sin(\eta) \end{bmatrix}$$

Identify the smallest elevation $h_{\min}$.
Sort the h-values, and identify the minimum h-value $h_{\min}$.

Expand V to $h_{\min}$.
Update V to represent the result of filling the puddle to a new height $h_{\min}$.

Return.

Figure 131 (continued): The *kinematic-BP*$_{e_{\text{ideal}}}$ procedure.

The control structure of the BP_e algorithm may be modified to take advantage of the incremental construction capabilities of the *CO* and *STATIC* algorithms. In this modification, the BP_e algorithm treats the configuration-obstacle surface $\mathcal{O}$ and possible-equilibrium states $\mathcal{Q}$ as lazy-evaluated data objects. This is accomplished by using the goal specification $\mathcal{G}$ to identify and construct the initial obstacle features, and then using the adjacency relationships of the resulting c-obstacle data structure to construct additional features as required by the *kinematic-*$BP_{e_{\text{ideal}}}$ procedure. There is a difficult subtlety involved in this process, described below. Special bookkeeping is required to assure that $\mathcal{Q}$ and $\mathcal{Q}'$ are properly maintained; these details are straightforward.

The procedures shown in Figures 129 through 131 reduce the problem of constructing a strong backprojection in the presence of uncertainty to the problem of identifying local minima and spillover points on the surface of the configuration obstacle, relative to a direction η. These critical points may correspond to a variety of topological situations on the obstacle surface, as shown in Figure 132.

When the algorithm contructs a set $\mathcal{M}$ of local minima or spillover points, each point $M \in \mathcal{M}$ must satisfy several criteria:

When constructing local minima:

- ▷ M must be a critical point on the obstacle surface, relative to the direction η.
- ▷ M must be a local minimum relative to η.

When constructing spillover points for a puddle V:

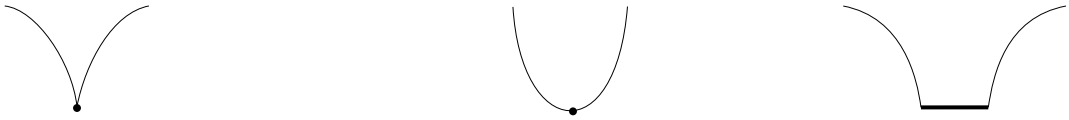
- ▷ M must be a critical point on the obstacle surface, relative to the direction η.
- ▷ M must be a saddle point or local maximum relative to η.
- ▷ M must be "relevant" to V; that is, as V is increasingly filled, M must touch the boundary of V at some point. This notion is illustrated in Figure 133.

These criteria suggest the following procedure for generating $\mathcal{M}$: Identify all critical points on the obstacle surface, and then classify them as local minima, maxima, or saddle points. If we are seeking local minima, then delete the local maxima and saddle points from $\mathcal{M}$, and return. If we are seeking spillover points, then delete the local minima from $\mathcal{M}$ and any remaining points that are not relevant to V, and return. This procedure raises the following questions:

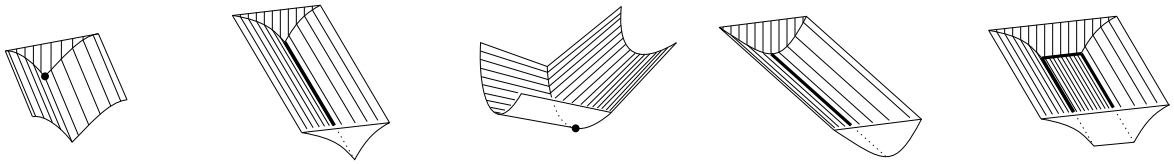
- How do we identify all critical points on the obstacle surface?
- How do we classify critical points as local minima, maxima, or saddle points?
- How do we decide whether a critical point is relevant to a puddle V?

Answers to these questions are sketched below.

(θ, y) local minima:

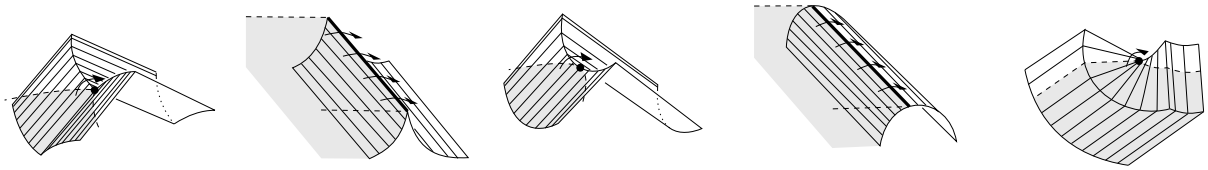


(x, y, θ) local minima:



(θ, y) spillover points:

(x, y, θ) spillover points:



(a) (b) (c) (d) (e)

Figure 132: Critical points on the configuration obstacle. The upper rows show local minima; extended sets must be bounded on all sides by features of increasing elevation with respect to η. The lower rows show spillover points; the dashed lines and shaded areas show puddles filled to the spillover point, while the arrows indicate the spill direction that would result from adding more water. The spillover points (a), (c), and (e) correspond to saddle points on the obstacle surface; (b) and (d) are local maxima. Bear in mind that in the three-dimensional examples, all applied force directions are defined in the (x, y) plane, and thus correspond to horizontal vectors in the (x, y, θ) space. In this figure, the (x, y, θ) axes have been re-oriented to provide a more intuitive viewpoint, with the applied force acting downward.

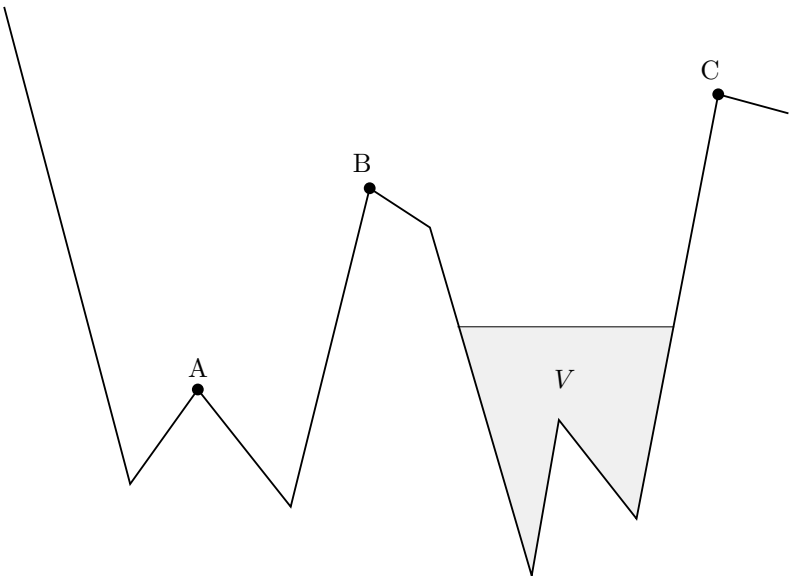


Figure 133: As water is added to the puddle V, it will spill past points B and C as the water level rises. Point A will be subsumed as the water level passes B. Points B and C are relevant spillover points; A is not.

Generating Critical Points

A point on the obstacle surface is critical whenever the set of inward-pointing normals on the surface contains the direction η (see Figure 134). This observation gives us a simple procedure for deciding whether a point M is a critical point: Construct the negated configuration-space friction cone for M, assuming zero friction. If the vector $[\cos(\eta) \quad \sin(\eta) \quad 0]^T$ is contained in the resulting force-sphere polygon, then M is critical.

This condition is exactly identical to the possible-equilibrium condition considered by the *STATIC* algorithm, assuming zero friction and zero applied-force uncertainty. Therefore we may generate all critical points on the obstacle surface by simply calling the *STATIC* algorithm, passing in a friction table $\mathcal{T}_0$ containing all zeros, a force direction interval $[\eta, \eta]$, and $r_\eta = 0$.

Classifying Critical Points

Given a critical point on the obstacle surface, we must decide whether it corresponds to a local minimum, saddle point, or local maximum. This is accomplished by forming the surface tangents at the critical point, and determining whether their dot-products with the vector $[\cos(\eta) \quad \sin(\eta) \quad 0]^T$ are all strictly negative, mixed, or strictly positive, respectively. This construction is illustrated in Figure 135. Care must be taken to properly handle the numerous special cases that can arise; some of these special cases are illustrated in Figure 136.

Figure 134: Determining whether a point on the obstacle surface is critical with respect to a direction η. (a) is critical; (b) is not.

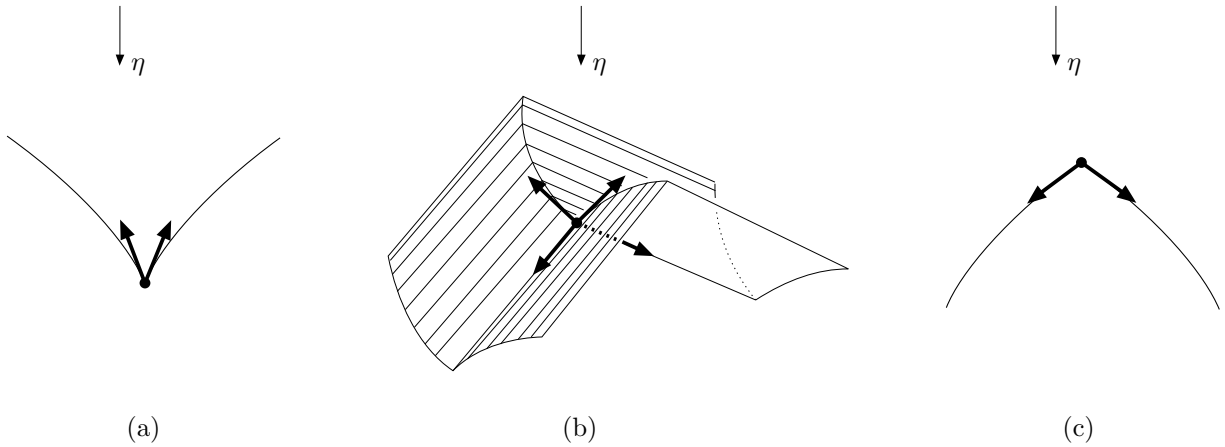


Figure 135: Classifying critical points. (a) A local minimum. (b) A saddle point. (c) A local maximum.

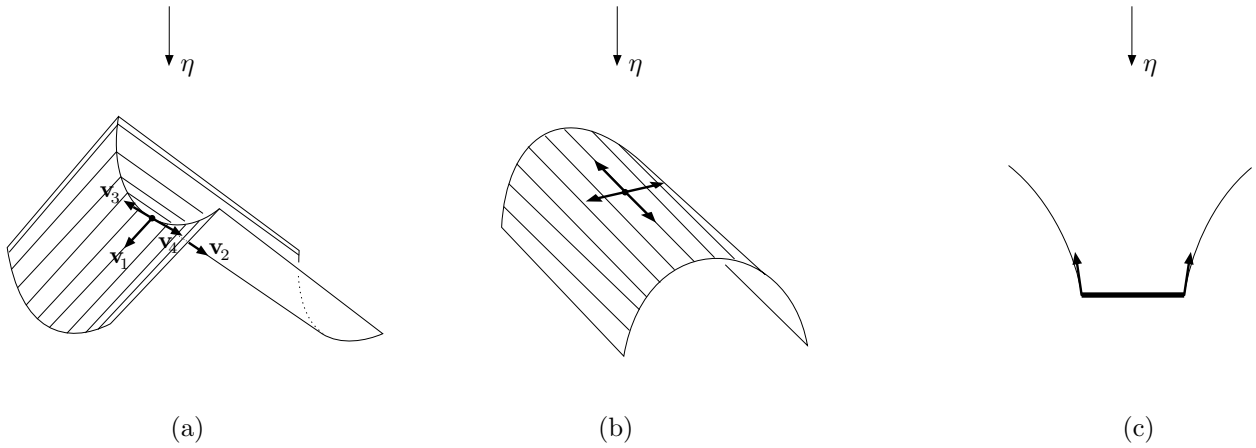


Figure 136: Example special cases. (a) The vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ must be generated on the adjacent facets to classify this saddle point. Further, since $\mathbf{v}_3$ and $\mathbf{v}_4$ are both perpendicular to η, a non-local or second-derivative analysis must be performed. (b) Since all tangent vectors are perpendicular to η, this local maximum must be analyzed using the original contact geometry, a non-local analysis, or a second-derivative analysis. Using the contact geometry is the simplest of these tests, and is already included in the *STATIC* algorithm's stability analysis. (c) Classifying this extended set of critical points requires examination of the obstacle features adjacent to the extended set.

Deciding Whether a Critical Point is Relevant

When the *kinematic-BP*$_{e_{\text{ideal}}}$ procedure constructs the maximum kinematic expansion of a puddle V, it must discriminate between relevant and irrelevant critical points (Figure 133). Relevant critical points may be recognized using the following decision procedure: Compute the elevation of the critical point with respect to the puddle surface and the direction η; if the critical point is not higher than the puddle surface, it is irrelevant. If the point is higher than the puddle surface, expand the puddle to the height of the critical point, forming V'. If the critical point touches the boundary of V', it is relevant; otherwise, it is irrelevant. This procedure is illustrated in Figure 137.

By applying the above procedure to all local maxima and saddle points on the obstacle surface, all spillover points relevant to the puddle V may be generated. However, this procedure is undesirable because it requires construction of the entire obstacle surface, and it requires an expensive puddle-expansion operation for every local maximum and saddle point. Can a more efficient procedure be found?

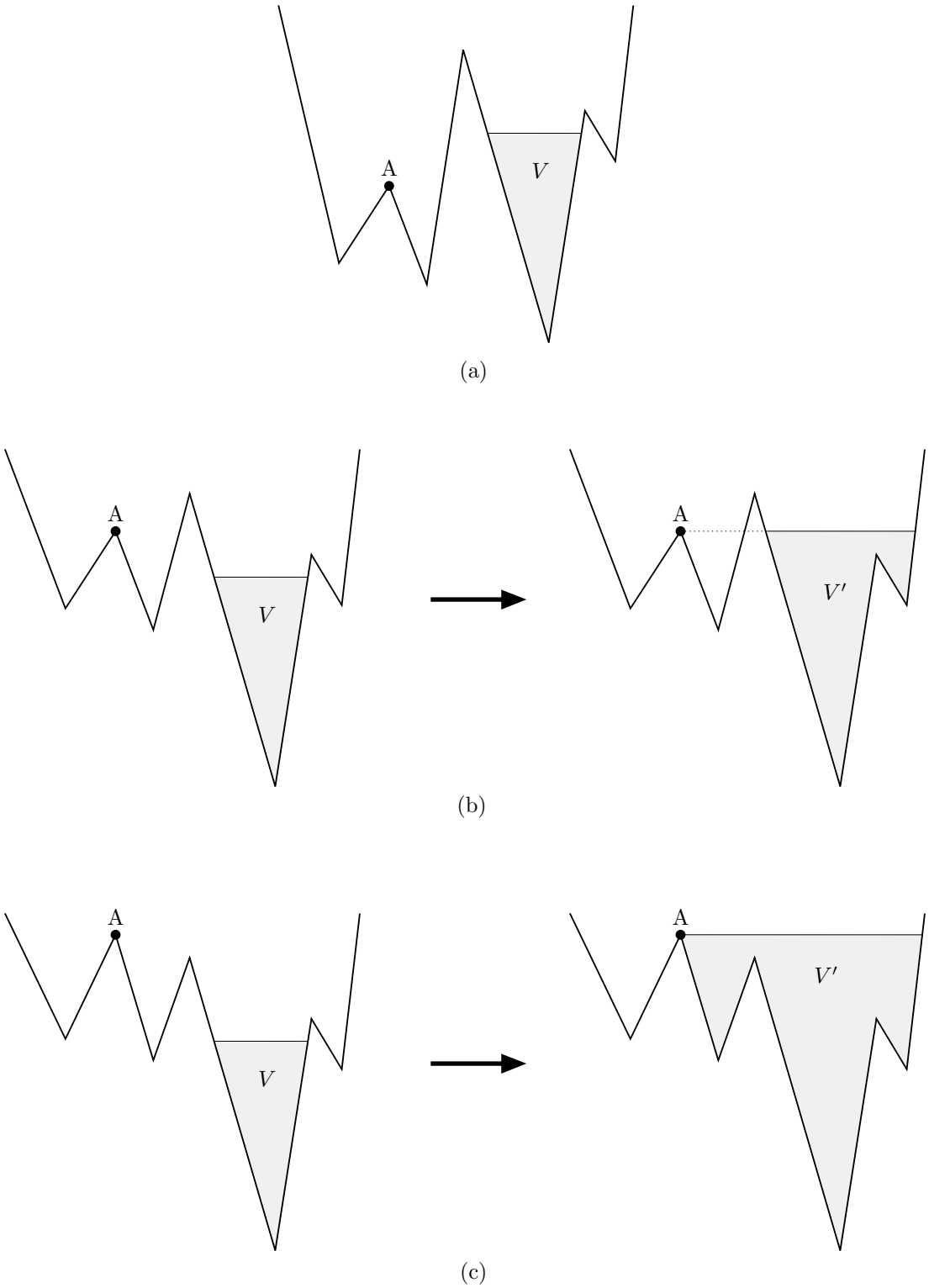


Figure 137: Deciding whether a spillover point is relevant to a puddle V. (a) The spillover point A is irrelevant because it is lower than the level of V. (b) The spillover point A is irrelevant because A does not touch the boundary of V', the result of expanding V to A's elevation. (c) A is relevant to V.

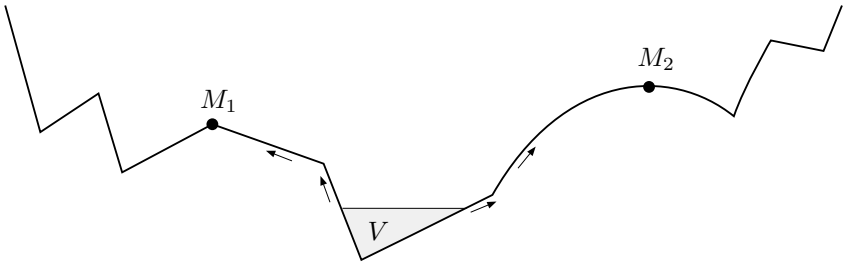


Figure 138: In this figure, the configuration-obstacle features adjacent to the boundary of the puddle V are traced until a local maximum is reached; the local maxima M_1 and M_2 nearest the puddle boundary are the only relevant critical points. The control structure of this search may be modified to stop the tracing procedure once the lowest critical point is found, reducing the number of obstacle features that must be analyzed.

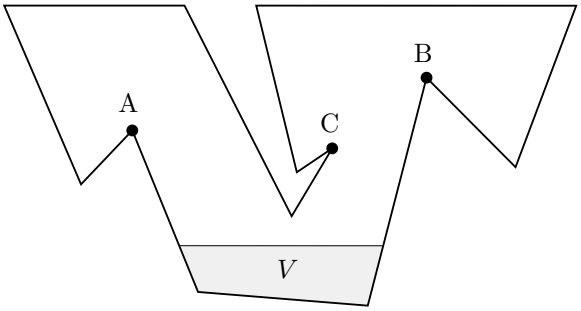


Figure 139: The tracing procedure illustrated in Figure 138 will identify A and B as relevant spillover points; the point C is missed.

One approach would be to use the features adjacent to the boundary of V to initiate a search for the nearest spillover point. This procedure is illustrated in Figure 138 for an example two-dimensional configuration space. An analogous procedure may also be defined for obstacles in the three-dimensional (x, y, θ) configuration space; this case is slightly more complicated because it is necessary to trace a graph of adjacent features instead of a linear sequence of features.

This procedure has the advantage that it supports incremental construction of the configuration obstacle. Since a partially-constructed c-obstacle data structure contains pointers connecting complete obstacle facets to their adjacent incomplete facets, the tracing procedure may call the *complete-facet* procedure to construct new obstacle features as they are needed.

Unfortunately, this procedure will fail to detect relevant spillover points in certain situations, such as the example shown in Figure 139. These failures occur because non-local features of the configuration obstacle pierce the puddle surface; this topological event is not detected by the tracing procedure. It may be possible to extend the tracing procedure to detect these cases, using decision procedures similar to the *conflict-possible?* tests employed by the *CO* algorithm. Further, fleshing out the geometric details of the puddle-expansion procedure may suggest other approaches. Defining a complete, efficient, and incremental algorithm for generating relevant spillover points remains an open research problem.

Critique

The BP_e algorithm may be used to construct strong backprojections for physical systems that have very complex dynamic properties. For example, the algorithm may be used to construct a set of initial dropping positions that will reliably locate a planar part in an orienting fixture, as shown in Figure 21 of Part I. The behavior of this physical system is dominated by inertial terms and collision dynamics, yet reliable actions may be synthesized without perfoming a detailed dynamic analysis. This is possible because the BP_e algorithm exploits the geometry of the task and the conservation of mechanical energy to identify invariants that apply to all possible system trajectories.

Care must be taken to insure that the assumptions required by the BP_e algorithm accurately match the physical system. For example, if we add vibration to the orienting fixture, a resonant excitation may occur that violates the half-space invariant (Figure 140). This problem may be avoided by applying the vibration along a direction normal to the support plane, or by performing a resonant-frequency analysis that demonstrates that anomalies such as the example shown in Figure 140 cannot occur. A third alternative might be to eliminate the vibration altogether by using an air table or magnetic levitation to eliminate support friction, or physically separating the fixture from the vibrating support plane.

The weak mechanics model employed by the BP_e algorithm enables it to construct strong backprojections for difficult problems involving complex shapes undergoing highly unpredictable trajectories. However, this same weak mechanics model prevents the algorithm from constructing backprojections for some very simple problems, such as the dropping task shown in Figure 141. The BP_e algorithm cannot construct a strong backprojection for this task, because the set of goal configurations does not correspond to a strict local minimum in the (x, y, θ) configuration space.

Expressed in the original space, this implies that the BP_e algorithm cannot bound the lateral motion of the polygon as it bounces on the horizontal surface; a series of bounces may cause the polygon to fall off the end of the surface (Figure 142). This is a plausible prediction for a highly elastic system, and the BP_e algorithm has no reason to rule out this possibility. Therefore the lack of a solution to this problem is not a shortcoming of the BP_e algorithm, but rather a shortcoming of the weak model of the task mechanics.

For some physical systems, we may know additional information about the system mechanics that further constrains the set of possible motions. For these systems, we would like to include this information in our backprojection construction procedure, thus producing a larger strong backprojection. This is the motivation for the BP_i algorithm, presented in the next section.

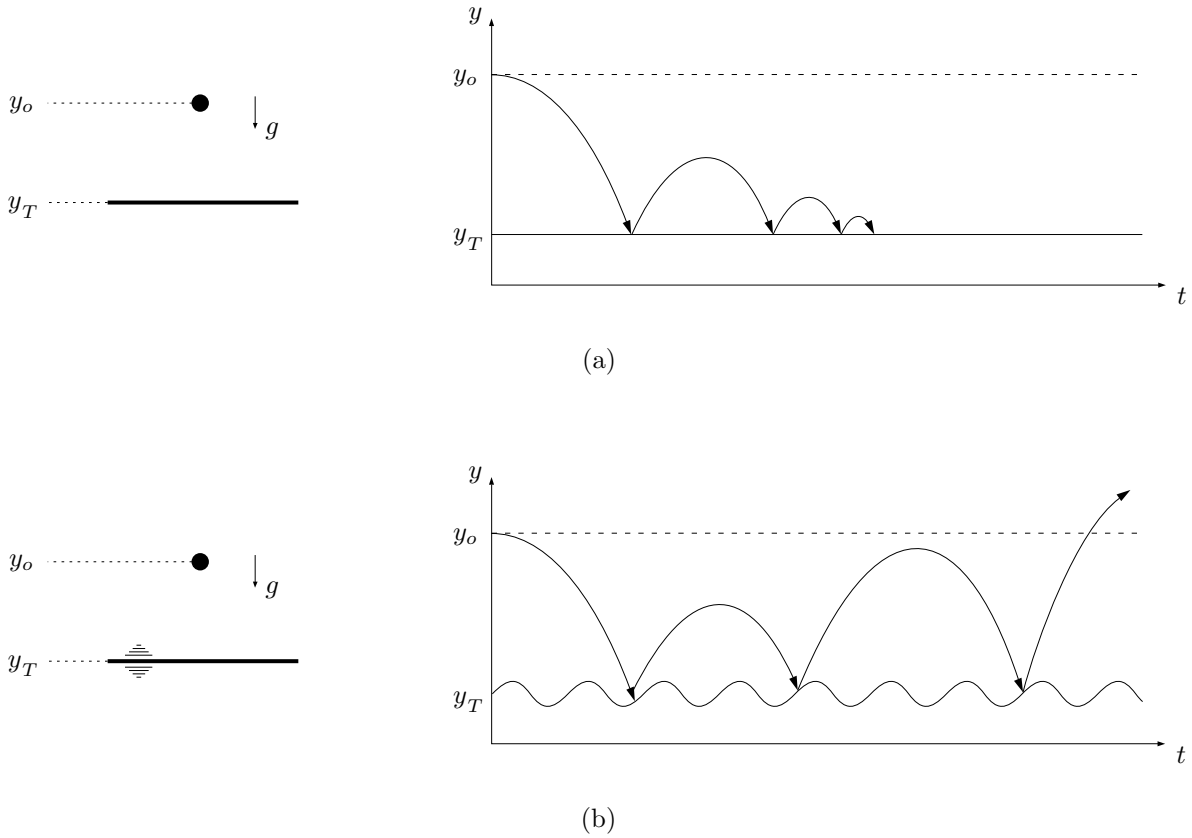


Figure 140: A semi-elastic particle dropping onto a vibrating table. (a) When the vibration is turned off, the bounce height decreases monotonically, and never exceeds the initial dropping height. (b) When vibration is added, resonant excitation may occur that causes the particle to exceed its initial dropping height. Here the vibration frequency is chosen so that the particle experiences a series of collisions that increase its energy.

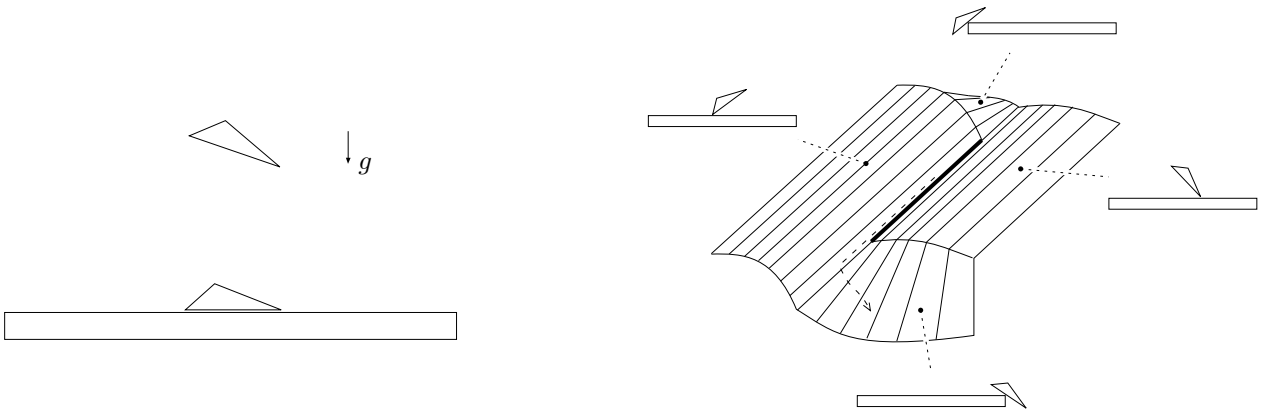


Figure 141: A simple task where the BP_e algorithm produces a null backprojection. (a) The goal is to drop the triangle so that it comes to rest on the flat surface in the orientation shown. (b) The task in the configuration space. The bold edge corresponds to the goal; this edge is not a local minimum because its endpoints are not adjacent to features with strictly increasing elevation. The dotted line shows a path in configuration space that exits the goal without increasing elevation.

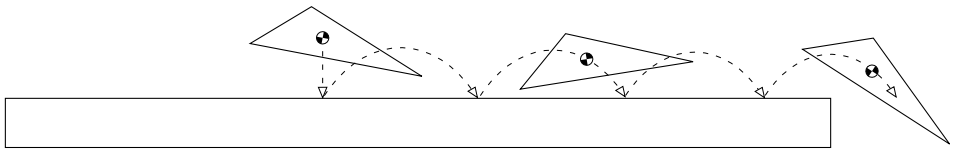


Figure 142: A trajectory demonstrating why the BP_e algorithm cannot produce a strong backprojection for the example shown in Figure 141.

Constructing V_{state} Using Numerical Integration

The BP_i algorithm constructs strong backprojections for physical systems where a model of the system mechanics may be used to constrain the set of possible instantaneous motions. Examples of such systems include objects being pushed along a flat surface, or objects controlled by a compliant-motion control law such as generalized damper or generalized spring control.

We model the differential behavior of these systems by a function D, which accepts a description of the current system state S and returns a set of states $\{S'\}$ that contains all possible states that may occur at the next time instant. D encapsulates a mechanical model of the system that includes all of the physical parameters needed to predict the system behavior, as well as uncertainty in those parameters.

To gain an intuitive understanding of the function D and its relationship to strong backprojections, consider the simple example of a particle sliding along a frictional constraint, shown in Figure 143. In this physical system, Coulomb friction opposes motion of the particle, and tends to bring the particle to rest. The behavior of this system may be expressed by a field of vectors in an $(x, \dot{x})$ state space (b). Each vector indicates the instantaneous change in state experienced by the particle; we refer to these vectors as *state-transition vectors.* Integrating these vectors forward from an initial state $(x_o, \dot{x}_o)$ will construct the set of states following the initial state (c). Similarly, reverse integration will construct the set of states preceding a given state (d). For this physical system, the function $D(x, \dot{x})$ returns the state-transition vector associated with the point $(x, \dot{x})$.

Now consider the same physical system, but with an uncertain coefficient of friction (Figure 144). This uncertainty causes the system behavior to become non-deterministic, producing cones of possible state-transitions in the state space (b). For each state-space point $(x, \dot{x})$, the associated *state-transition cone* indicates the set of possible states that may occur at the next time instant. Forward-integration may again be applied from an initial state $(x_o, \dot{x}_o)$; the resulting region contains all states that may possibly follow the initial state (c). A similar reverse-integration process identifies the set of states that may possibly precede a given state (d). With uncertainty present, the function $D(x, \dot{x})$ returns the state-transition cone associated with the point $(x, \dot{x})$, instead of a single state-transition vector.

The state-transition cone function D may be used to construct strong backprojections, as shown in Figure 145. In this figure, $\mathcal{G}$ represents a set of goal configurations, and the extrema of the state-transition cones are reverse-integrated to identify a set of initial states that will reliably achieve a final configuration in $\mathcal{G}$. Constraints may also be included, as shown in Figure 145(b).

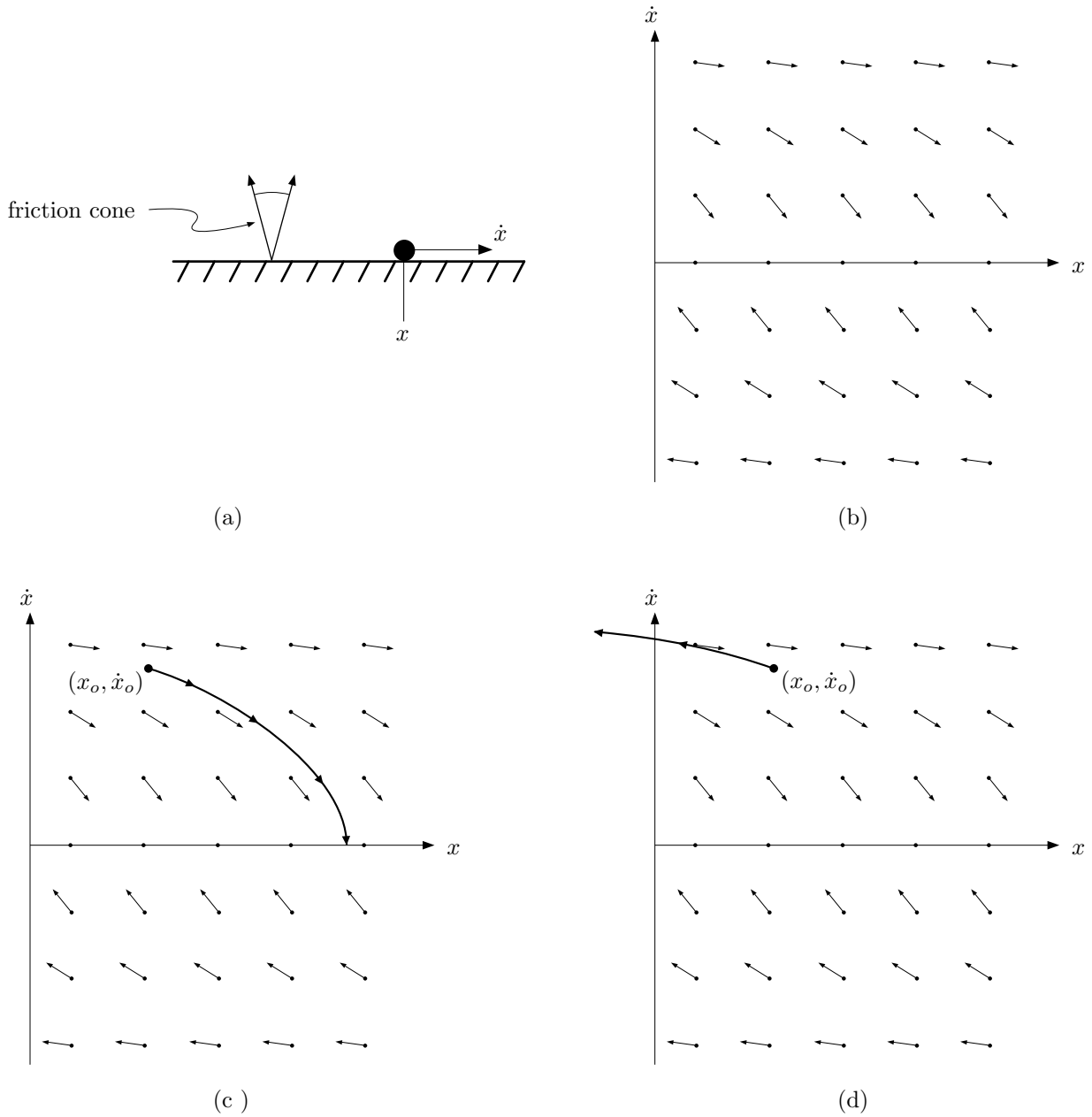


Figure 143: A simple physical system. (a) A particle in contact with a frictional constraint. (b) The field of state-transition vectors describing the particle's behavior, expressed in the $(x, \dot{x})$ state space. (c) Forward integration of the state-transition field. (d) Reverse integration.

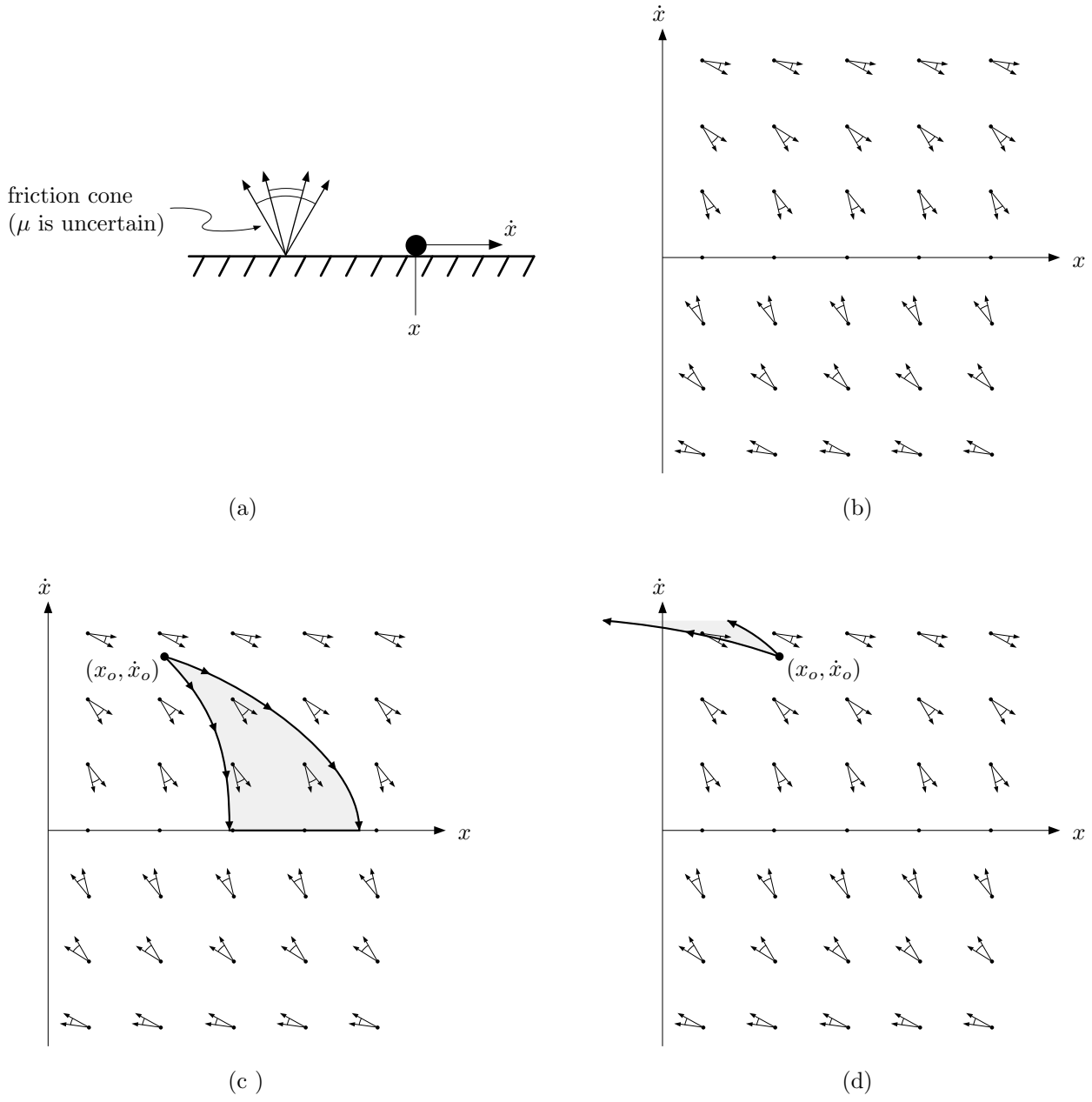


Figure 144: Adding uncertainty. (a) The physical system of Figure 143, but with uncertainty in the coefficient of friction. (b) The field of state-transition cones that describe the particle's behavior. (c) Forward integration. (d) Reverse integration.

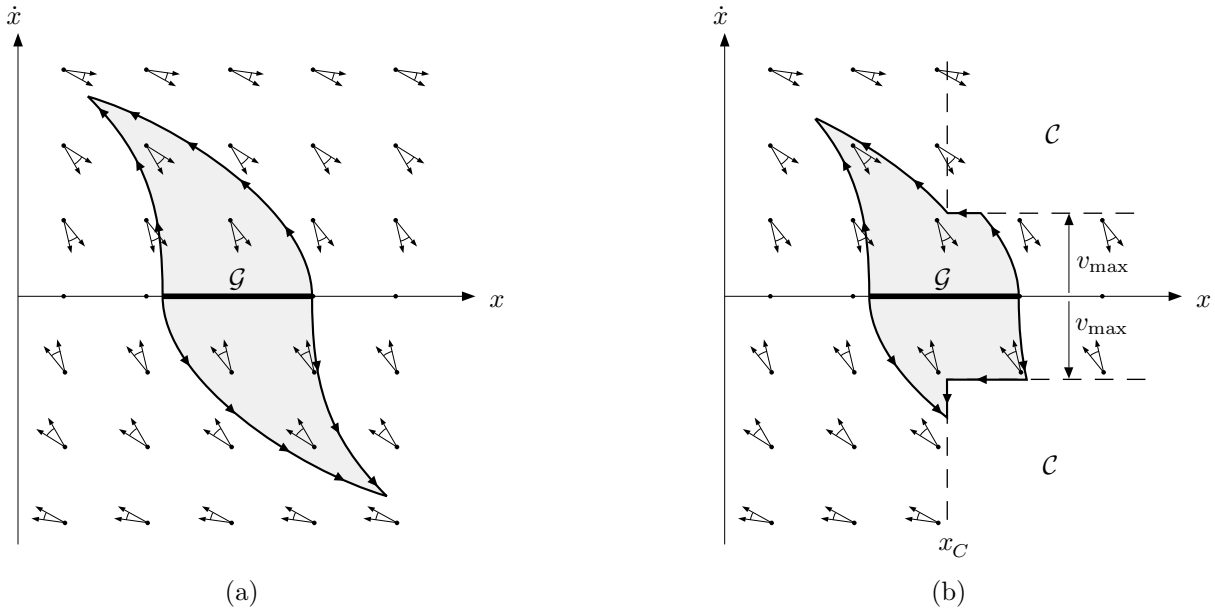


Figure 145: Using the state-transition field to construct strong backprojections. (a) The goal is to place the particle at rest within a desired interval of x-values; this is indicated by the bold segment $\mathcal{G}$. Reverse-integration is performed from the endpoints of $\mathcal{G}$, delineating the shaded region. This region corresponds to the set of initial states that will reliably achieve $\mathcal{G}$, despite uncertainty in the physical system. (b) The same task, but with the artificial constraint that the particle's speed should never exceed $v_{\max}$ whenever x is larger than x_C.

The example shown in Figure 145 suggests a procedure for constructing strong backprojections: Given a goal $\mathcal{G}$, reverse-integrate the extrema of the state-transition cones from the endpoints of $\mathcal{G}$, using the function D to generate the necessary state-transition cones. If a constraint-set $\mathcal{C}$ is encountered, trace around the boundary of $\mathcal{C}$ in the appropriate direction until integration may resume. Terminate this process when the resulting contours cross, enclosing a connected region of initial states that will reliably achieve $\mathcal{G}$.

This procedure is conceptually simple, but unfortunately will not properly construct backprojections for general tasks. There are four reasons for this:

- *Not all goals have endpoints.* Some goals are regions that do not have endpoints appropriate for starting the integration process. Figure 146 shows an example.
- *Enclosed constraint regions may be overlooked.* The proposed procedure has no mechanism for detecting constraint states that are enclosed within the integrated contours. This may lead to the construction of semantically incorrect backprojections (Figure 147).
- *The procedure may not terminate.* Some backprojections are unbounded; the bounding contours of these backprojections will never cross, causing the numerical integration to proceed indefinitely. Figure 148 shows an example.
- *The procedure is prone to numerical errors.* Small numerical errors in the state-transition cone may be amplified by the integration process, leading to gross errors in the resulting strong backprojection (Figure 149).

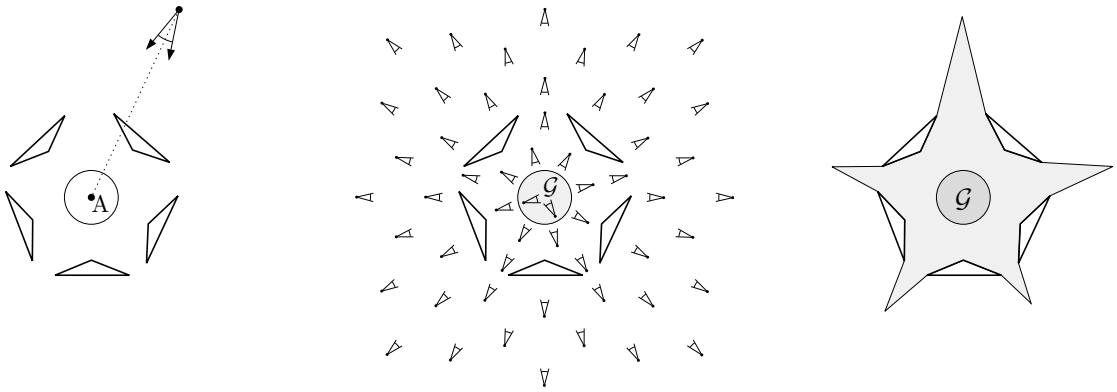


Figure 146: An example of a goal that has no endpoints for initiating reverse integration. (a) A physical system where a point robot moves toward the point A, with direction error. The goal is to place the robot inside the circle. (b) The situation in the robot's configuration space. The goal is the circular disc $\mathcal{G}$. (c) The set of configurations from which the robot will reliably reach $\mathcal{G}$.

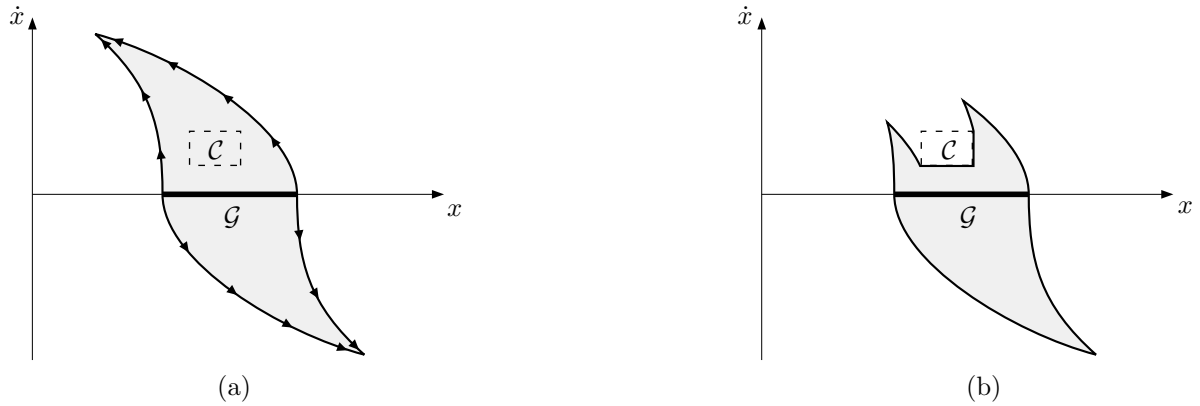


Figure 147: The previous sliding-particle example, with a bounded constraint set $\mathcal{C}$. (a) The proposed reverse-integration procedure inadvertantly encloses $\mathcal{C}$, producing an erroneous backprojection. (b) The correct strong backprojection.

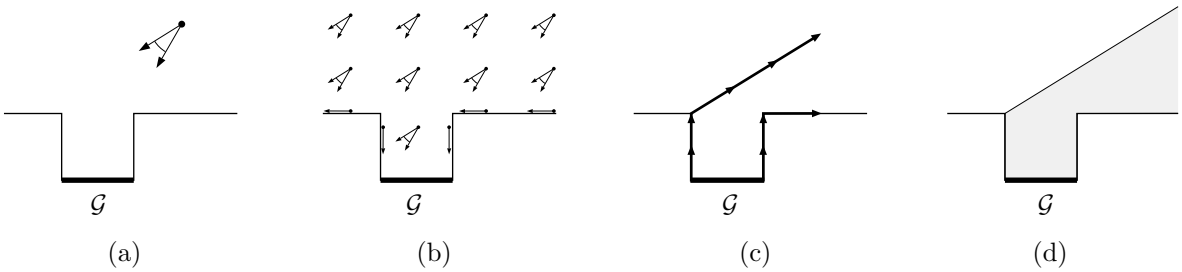


Figure 148: (a) A classic peg-in-hole task, due to [Lozano-Pérez *et al.* 1984]. (b) The state-transition field of this physical system. (c) The proposed reverse-integration procedure fails to terminate in this example, since the integrated contours never cross. (d) The region of initial configurations that will reliably achieve $\mathcal{G}$ is unbounded; any subset of this region is a valid strong backprojection.

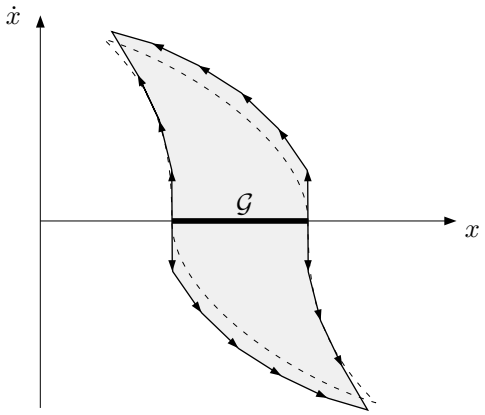


Figure 149: The finite precision of numerical integration may cause the proposed reverse-integration procedure to include points outside the true backprojection (shown dotted).

It may be possible to modify the above procedure to overcome these problems, but instead we will consider another procedure that avoids these difficulties. This procedure constructs a contour surrounding the goal $\mathcal{G}$, and iteratively expands the contour to include adjacent points whose state-transition cones strictly enter the contour. This construction is illustrated in Figure 150.

The BP_i algorithm is based on this contour-expansion procedure. The algorithm constructs strong backprojections for physical systems involving two polygonal objects whose interaction can be described by a state-transition cone function $D(x, y, \theta, \text{feature})$; this allows strong backprojections to be constructed in the (x, y, θ) configuration space. The BP_i algorithm constructs and expands contours on the surface of the configuration obstacle, and uses a ray-construction procedure to lift the resulting two-dimensional surface patches into three-dimensional configuration-space volumes. The ray-construction procedure further restricts the class of physical systems that may be addressed by the BP_i algorithm: All free-space motions must be pure translations, in a direction within a constant bounding interval.

The following sections describe the BP_i algorithm and its data structures; the chapter concludes with a discussion of ways to extend the BP_i algorithm to address a broader class of tasks.

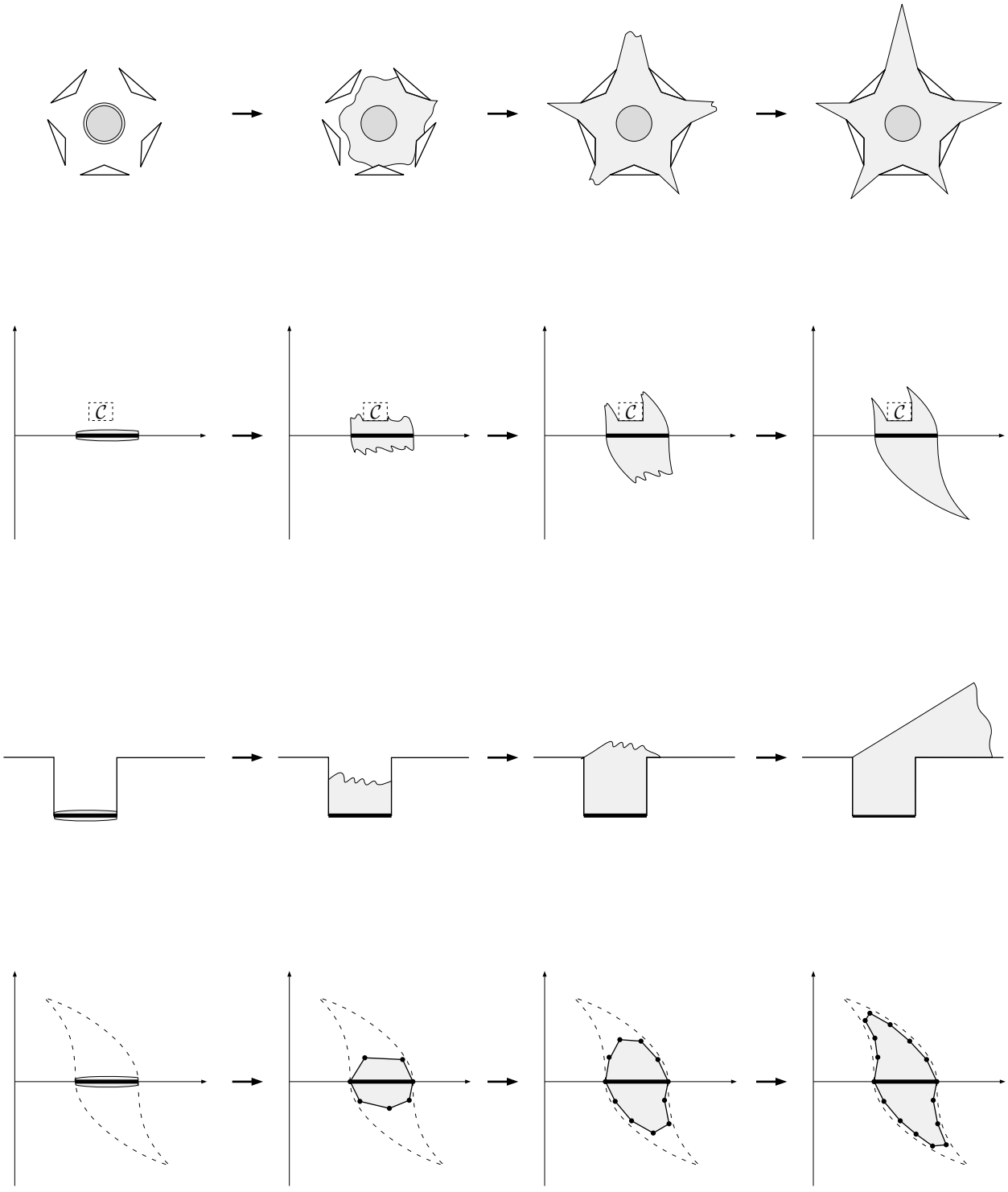


Figure 150: The contour-expansion procedure applied to the examples of Figures 146–149.

Representing State-Transition Cones, Expandable Contours, and Strong Backprojections

The BP_i algorithm uses three special data structures in its computation of strong backprojections. These data structures represent state-transition cones in the (x, y, θ) configuration space, expandable contours on the surface of a configuration-space obstacle, and volumes of (x, y, θ) configurations corresponding to the strong backprojection V_{state}. This section will describe these data structures and some of the basic operations for manipulating them.

State-Transition Cones

The function $D(x, y, \theta, \text{feature})$ constructs the state-transition cone for the configuration (x, y, θ); this cone delineates the configurations that may occur in the time instant following (x, y, θ). The state-transition cone may also be viewed as a description of the set of possible differential motions the moving-object may undergo relative to the fixed-object. This set may include a collection of free-space motions, a collection of motions on the configuration-obstacle surface, no motion, or any combination of these.

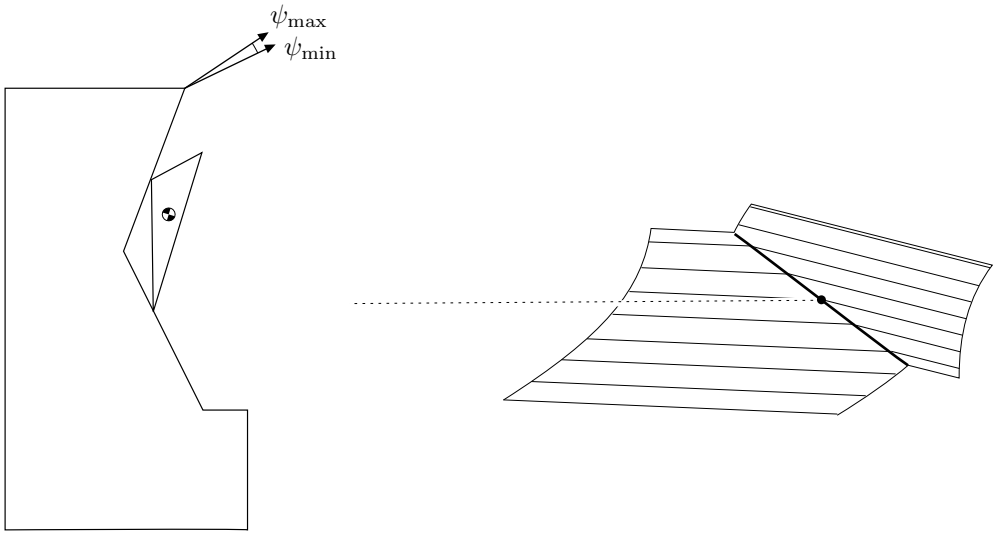
A data structure for representing state-transition cones is shown in Figure 151. This data structure contains fields for each of the motion classes mentioned above; the set of possible motions is the union of these classes. The data structure may be used to represent general state-transition cones, under the restriction that free-space motions must be translations.

The state-transition cone data structure is constructed by the function $D(x, y, \theta, \text{feature})$. This function is specific to the physical system being analyzed, and considers a variety of physical parameters describing the interacting objects and the applied action. Figure 152 briefly summarizes the construction of state-transition cones for systems involving linear pushing actions.

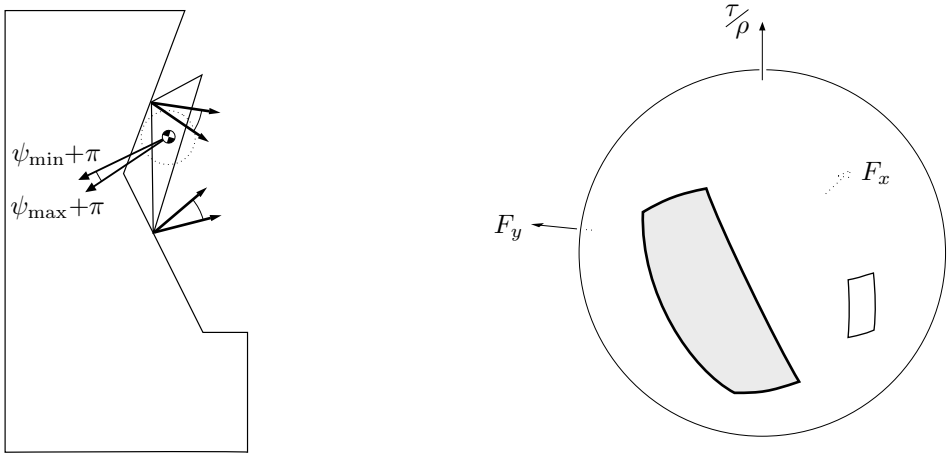
state-transition-cone

no-motion?	A Boolean flag indicating whether static equilibrium is possible for the configuration (x, y, θ).
edge-motions	If the point (x, y, θ) is on a configuration-obstacle edge or vertex, then this is a set of $\langle E$ plus? minus?$\rangle$ data records. Each record is comprised of an obstacle edge E, and plus? and minus? flags indicating the possible motion directions along E.
facet-motions	If the point (x, y, θ) is on the configuration-obstacle surface, then this is a set of $\langle F\ \{[\beta_{\min_1}, \beta_{\max_1}], [\beta_{\min_2}, \beta_{\max_2}], \ldots\}\rangle$ data records, each containing an obstacle facet F and a set of angular intervals $\{[\beta_{\min_1}, \beta_{\max_1}], [\beta_{\min_2}, \beta_{\max_2}], \ldots\}$, that delineate a set of possible motions on the surface of F. These beta-values are defined in the $(p, \rho\theta)$ space that parameterizes F's c-surface, where ρ is a scaling factor chosen to improve numerical robustness.
free-motions	This is a set of intervals $\{[\eta_{\min_1}, \eta_{\max_1}], [\eta_{\min_2}, \eta_{\max_2}] \ldots\}$, representing the set of possible free-space translation directions, measured relative to the fixed-object's x-axis.

Figure 151: A data record for representing the state-transition cone corresponding to a configuration-space point (x, y, θ).



(a)



(b)

Figure 152: Constructing the state-transition-cone for linear pushing actions. This figure shows the execution of the $D(x, y, \theta, \text{feature})$ function for an example configuration on an obstacle edge. This function creates a state-transition-cone data record, and performs a physical analysis to fill the record's no-motion?, edge-motions, facet-motions, and free-motions data fields appropriately. (a) The example configuration. The pushing direction interval $[\psi_{\min}, \psi_{\max}]$ is built into the function, and the other required physical parameters are built into the moving-object and fixed-object data structures, which are accessible through the configuration-obstacle feature. (b) Determining whether "no-motion?" is possible. Linear pushing satisfies the assumptions required by the *STATIC* algorithm, so the D function applies the possible-equilibrium test described in the Statics chapter. In this case equilibrium is not possible.

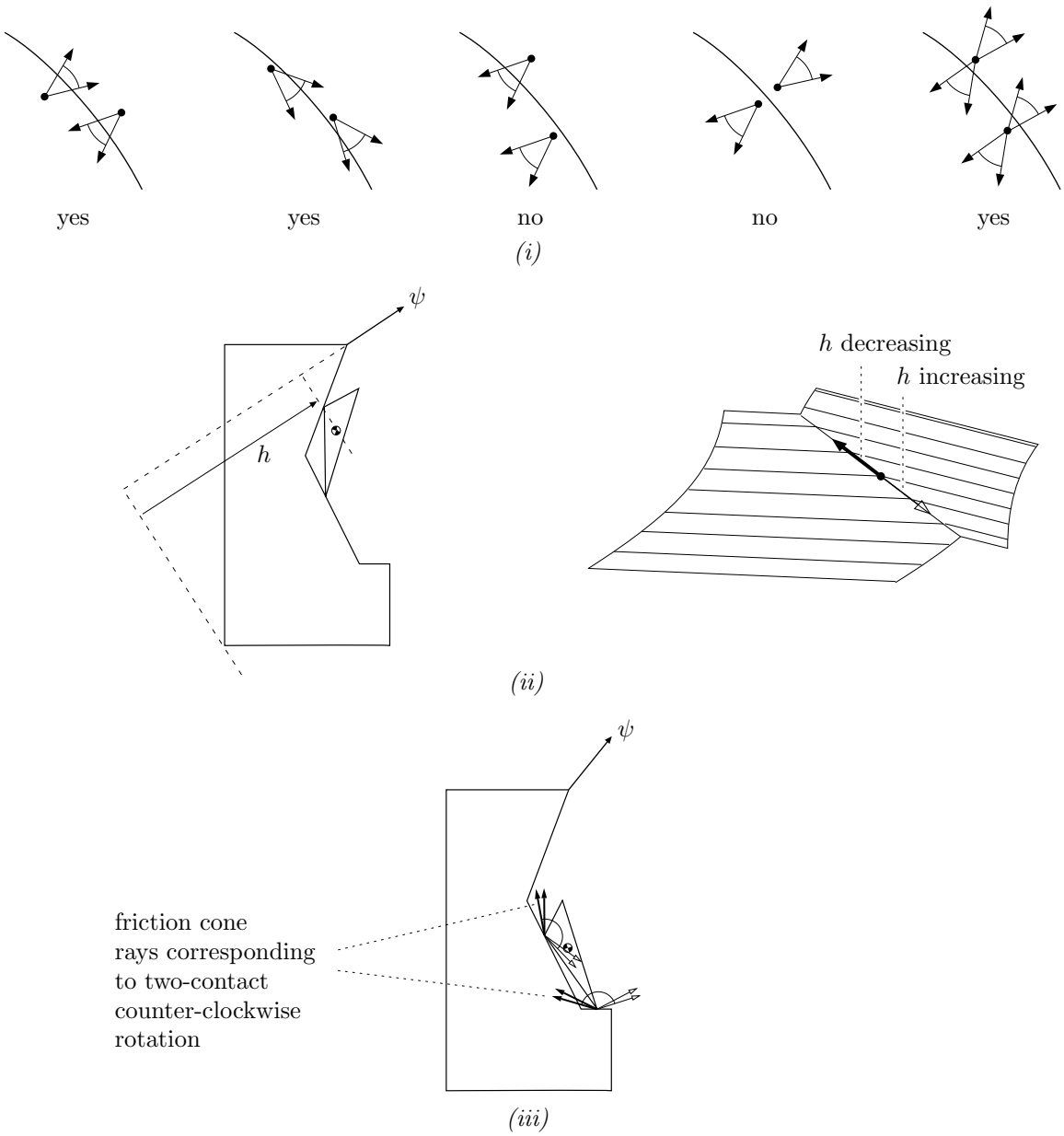


Figure 152 (continued): (c) Constructing the set of possible edge-motions. The D function decides whether the configuration may remain on the edge, identifies the motion direction along the edge, and then checks the consistency of this motion. (*i*) To decide whether the configuration may remain on the edge, the function constructs the state-transition-cones of the adjacent facets, and concludes that configurations on the edge may persist if both adjacent state-transition-cones include a motion toward the edge. (*ii*) To decide the motion direction, the function invokes the conjecture that the center of friction never rises relative to the pushing direction. Put another way, the signed distance h never increases. This conjecture uniquely determines the motion direction along the edge, although uncertainty in ψ may cause ambiguities. (*iii*) In some cases the function is able to determine that motion in a given direction is not possible, based on the moment exerted by the contact friction cone rays. In the example shown, counter-clockwise motion is consistent with the h conjecture, but is inconsistent with the clockwise moment exerted by the corresponding friction-cone rays.

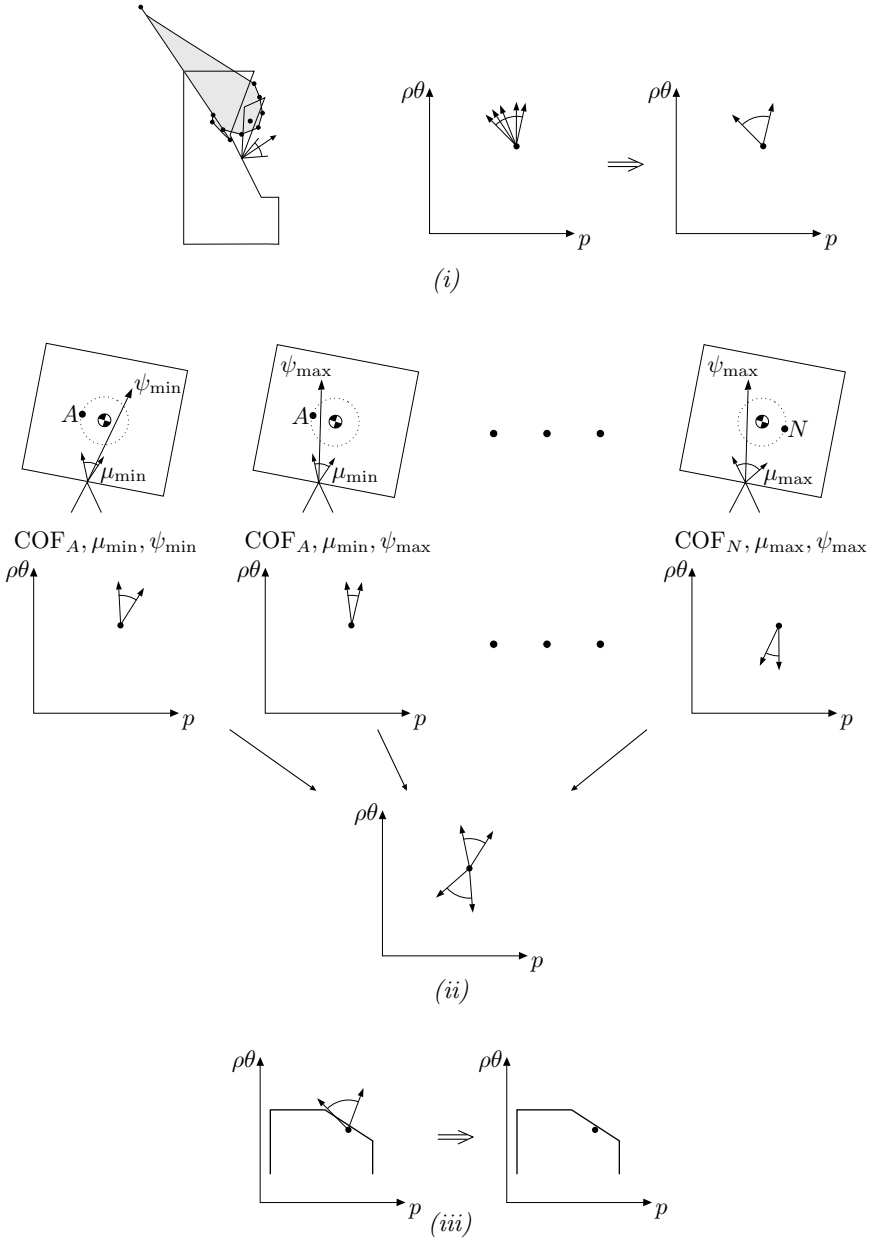


Figure 152 (continued): (d) Constructing the set of possible facet-motions. The D function constructs the set of possible motions for each facet adjacent to the obstacle edge, and forms the union of these sets. The analysis of a single facet is shown here. (*i*) Given a pushing situation without uncertainty, the function constructs an approximation to the COR locus developed by Peshkin. This is the set of possible instantaneous rotation centers, given that the distribution of support points beneath the moving-object is unknown but contained within the object boundary. The vertices of this approximation are transformed into instantaneous motion directions in the facet's $(p, \rho\theta)$ space, and the convex span of these directions is formed. (*ii*) Uncertainty is included by applying this construction to all combinations of the uncertainty bounds, including contact friction, pushing direction, and center-of-friction location. The resulting cones are then combined to form the set of motions possible on the facet c-surface, given these sources of uncertainty. This process must be implemented carefully; the combination of two cones is not simply their convex span. (*iii*) Kinematic constraints are included by removing motion directions that exit the facet; these are identified using the tangent direction of the obstacle edge, expressed in the facet's $(p, \rho\theta)$ space.

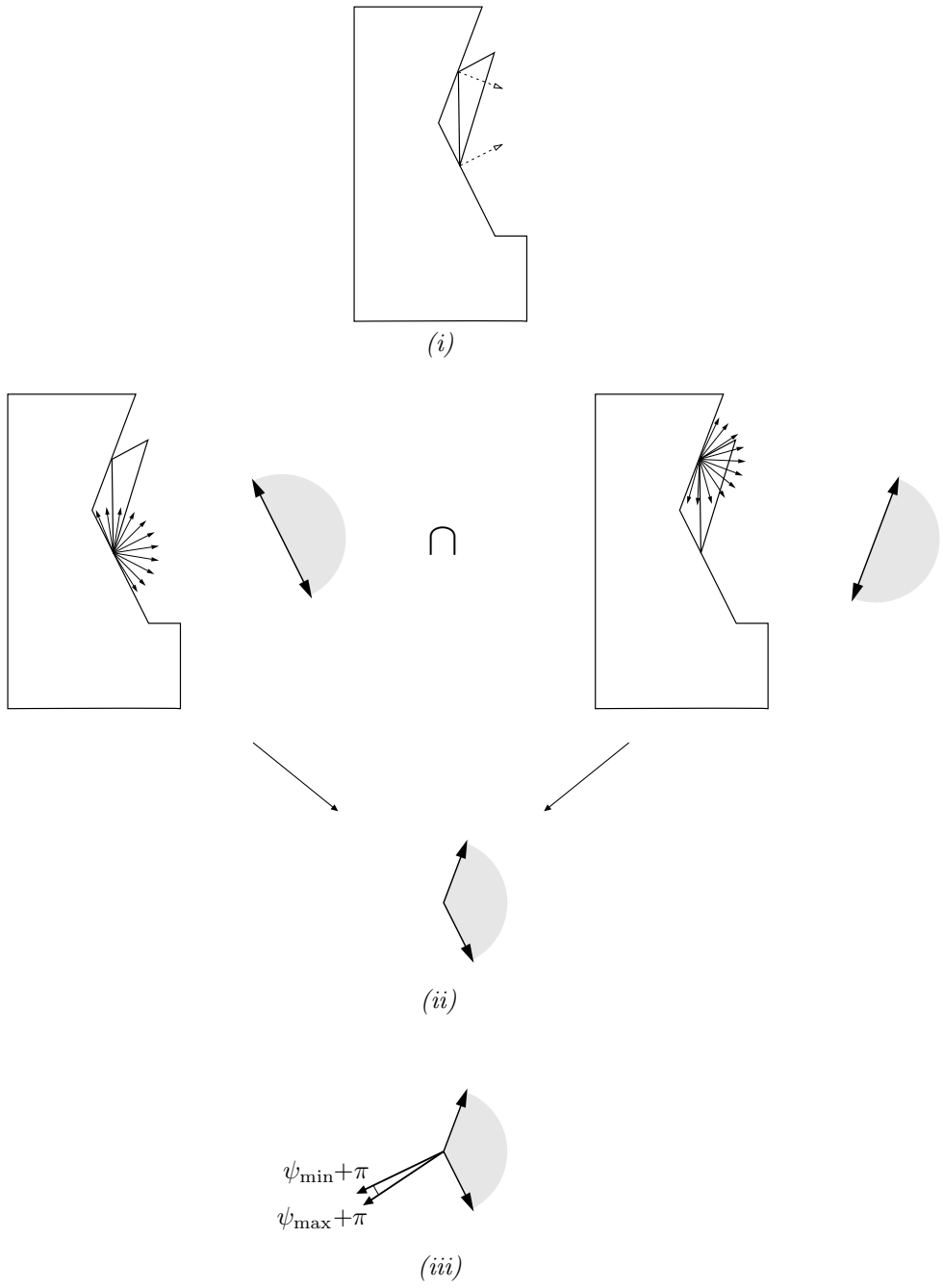


Figure 152 (continued): (e) Constructing the set of possible free-motions. (*i*) The D function constructs the contact normals for the configuration. (*ii*) For each contact normal, the function identifies the set of free-space translation directions consistent with the contact normal. The resulting intervals are intersected to form the set of free-space translation directions that are consistent with all kinematic constraints. (*iii*) The set of possible pushing directions is negated and intersected with the set of kinematically admissible free-space translations; the result of this intersection is the set of possible free-motions. In this case, no free-motions are possible. This analysis of linear pushing actions draws heavily on the work of other authors; see [Reuleaux 1876], [Mason 1986a], [Peshkin and Sanderson 1988a], and [Goyal 1989] for further information.

Expandable Contours

The primary integration technique employed by the BP_i algorithm is the expansion of closed contours on the configuration obstacle surface. These expandable contours are represented by a circular doubly-linked list of contour-point data records, shown in Figure 153.

These contours are iteratively expanded by visiting each contour-point, determining whether the point can be expanded, and re-setting the (x, y, θ) coordinates of the contour-point to a new expanded position (x', y', θ'). This process is applied to every contour-point, and repeated until no further expansion is possible, or some other termination condition is satisfied. The construction of new expansion points will be described in a later section; here we will review the basic expansion operations necessary to maintain the integrity of the contour data structure.

As the contour is constructed and expanded, the BP_i algorithm maintains the following invariants:

- ▷ Every obstacle feature crossed by the contour is explicitly represented in the contour data structure. This implies that a contour-point exists for every obstacle vertex that touches the contour, and at least one contour point exists for every obstacle edge and facet that the contour traverses.
- ▷ The contour never crosses itself.
- ▷ The distance between contour-points never exceeds some pre-set threshold $l_{\max}$ in x and y, or $\lambda_{\max}$ in θ. In a sequence of contour-points traversing a single obstacle edge or facet, no two points are simultaneously closer than $l_{\min}$ and $\lambda_{\min}$.

These invariants are initially assured when the contour is constructed, and then maintained during all subsequent expansion operations. Figure 154 shows an example contour satisfying these invariants, and Figures 155 through 158 illustrate a few of the procedures employed to maintain these invariants as the contour is expanded. The second invariant is automatically maintained as an artifact of the strong-backprojection process; special procedures are not required to prevent the contour from crossing itself during contour-expansion.

contour-point

x y θ	The coordinates of this contour-point.
type	One of `facet`, `edge`, `vertex`, or `interface`.
F E V	Obstacle features adjacent to this contour-point. If the type is `interface`, then F and E are both defined.
previous next	Adjacent contour-points in the contour. The orientation of the contour is significant; if you walk on the obstacle surface from the previous contour-point to the next contour-point, points inside the contour are on your left.

Figure 153: A data record for representing a point on an expandable contour; some control fields are omitted for clarity.

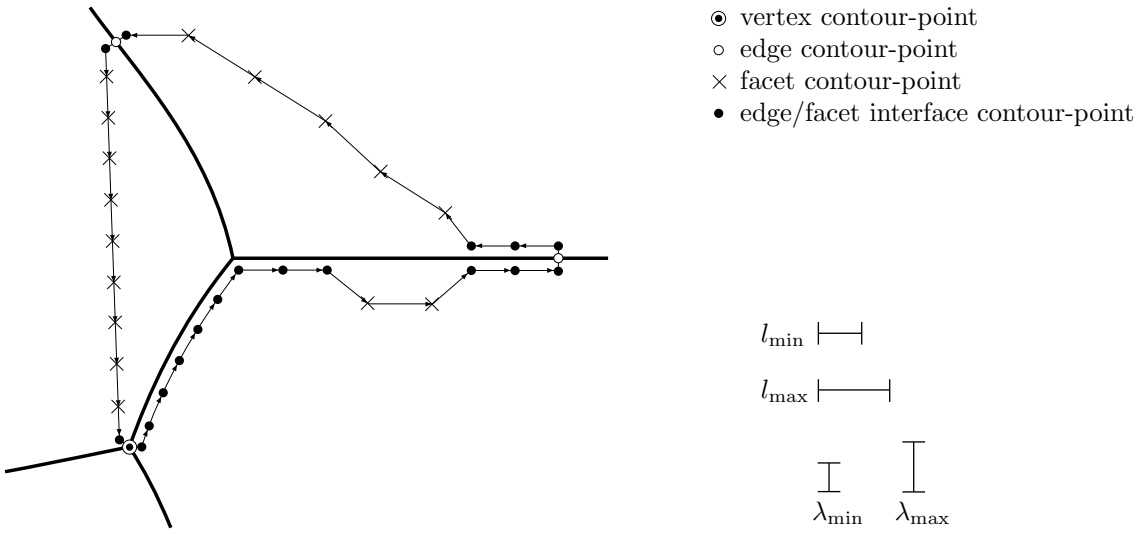


Figure 154: An example contour, containing all types of contour-points. The edge/facet interface contour-points are drawn slightly offset for clarity; their actual position is exactly on the associated obstacle edge. All contour-segments are within the $l_{\max}$ and $\lambda_{\max}$ bounds, and almost all contour-segments segments obey the $l_{\min}$ or $\lambda_{\min}$ bounds. Exceptions occur when contour-points are very close to another obstacle feature, such as the obstacle vertex at the lower left. Zero-length contour-segments always connect edge or vertex contour-points with their adjacent edge/facet interface contour-points.

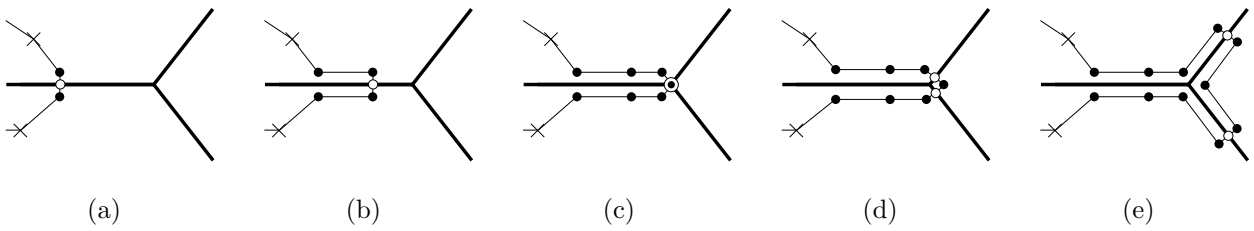


Figure 155: Expanding a contour-point past an obstacle vertex. (a) An initial contour. (b) A normal expansion step. (c) The next expansion point exceeds the edge-limits, so the contour-point stops at the obstacle vertex. (d) Expanding the obstacle vertex is a purely topological change in the contour; the collection of five edge and edge/facet interface contour-points near the vertex are all collocated. (e) After passing the vertex, expansion proceeds normally. In this figure, the facet and edge/facet interface contour-points are left static for clarity.

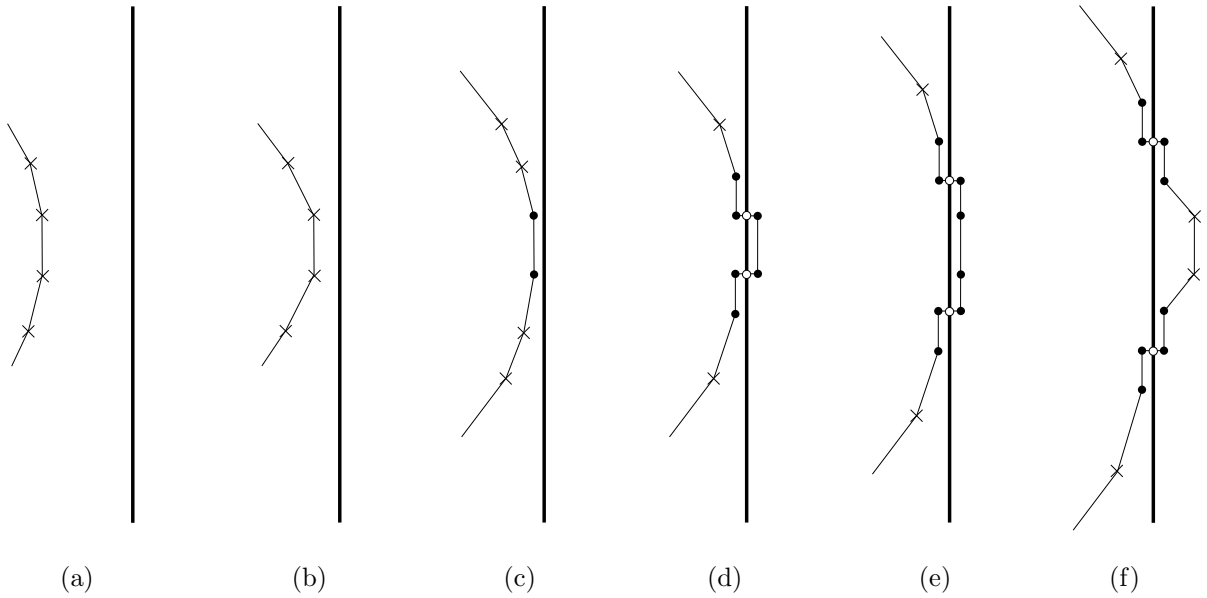


Figure 156: Expanding a contour past an obstacle edge. (a) An initial contour on an obstacle facet. (b) A normal expansion step. (c) Further expansion of the middle two contour points would pass the edge bounding the facet, so these points stop at the edge. (d–f) Crossing the edge, allowing normal expansion to resume on the opposite side.

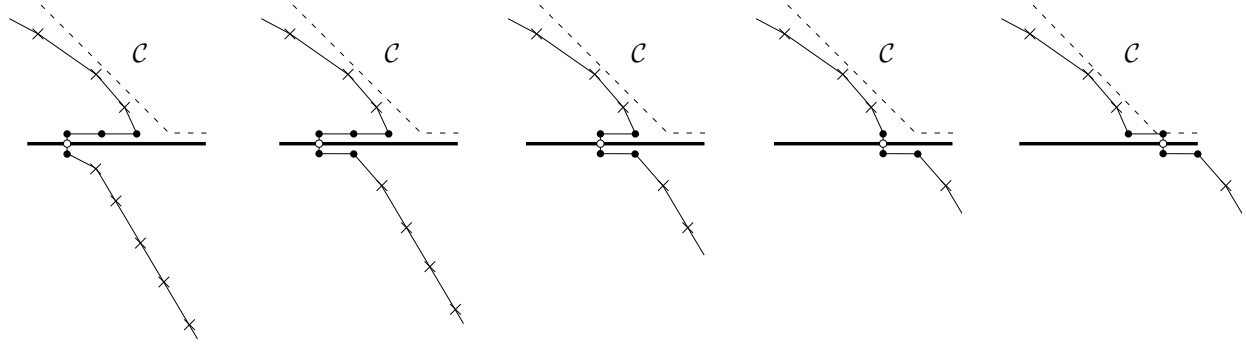


Figure 157: Managing edge/facet interface contour-points. In this example, the upper facet contour-points have expanded more quickly than the contour-points on the lower facet, but their expansion is halted by the constraint set $\mathcal{C}$. The contour-points on the lower facet continue to expand, overtaking the edge/facet contour-points on the upper facet; these are deleted as they are overtaken.

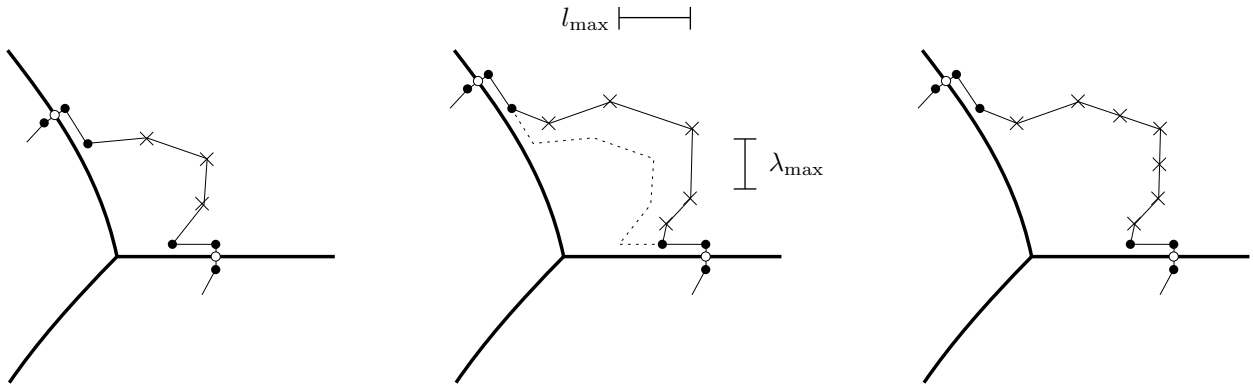


Figure 158: Maintaining the maximum-length invariant. (a) An initial contour. (b) After expanding all contour points; the previous location of the contour is shown dotted. Two of the expanded contour-segments exceed the length bounds $l_{\max}$ and $\lambda_{\max}$; these segments are split, as shown in (c). The minimum bounds $l_{\min}$ and $\lambda_{\min}$ are also maintained; this is not shown.

Strong Backprojection Volumes

Once a contour has been expanded, the BP_i algorithm uses a ray-construction procedure to convert the enclosed two-dimensional surface patch into a three-dimensional strong backprojection V_{state}. This volume is represented by a collection of θ-slices that approximate the true volume. An (x, y, θ) point is considered to be within the volume if it is contained within a slice with a matching θ-value, or if it is contained within both of the nearest θ-slices. This representation is similar to slice-based representation described in [Lozano-Pérez 1987]. Figure 159 shows a data structure for representing approximate volumes in configuration space.

c-volume

θ-interval	The set of θ-values spanned by this volume. This may be a closed interval $[\theta_{\min}, \theta_{\max}]$, or an "all-angles" flag indicating that the volume spans all θ-values.
slice-list	An ordered list of c-slices, spaced at regular θ-values.

c-slice

theta	The θ-value for this slice.
polygons	A set of polygons defining the intersection of the volume with this θ-plane.

Figure 159: Data records for representing volumes in configuration space.

The BP_i Algorithm

The BP_i algorithm accepts the following input:

- A polygonal moving-object $\mathcal{P}_m$.
- A polygonal fixed-object $\mathcal{P}_f$.
- A function $D(x, y, \theta, \text{feature})$, which returns a state-transition cone bounding the possible motions that may occur from the configuration (x, y, θ) on the indicated configuration obstacle feature. D may assume that it is never given an illegal configuration, but it should be defined for free-space points as well as points on the obstacle surface.
- A set of desired goal configurations $\mathcal{G}$.
- A set of task constraints $\mathcal{C}$, defining a set of configurations that should never occur.
- A predicate *stop*?(contour), which may be used to terminate contour expansion early if desired.

Given this input, the algorithm constructs a set of configurations V_{state} that will reliably achieve a final configuration in $\mathcal{G}$, never encountering a configuration in $\mathcal{C}$. The BP_i algorithm constructs V_{state} by constructing an initial contour surrounding the stable subset of $\mathcal{G}$, and then expanding this contour to include additional points outside $\mathcal{C}$ that strictly enter the contour. This contour-expansion may be interrupted at any time to lift the enclosed two-dimensional region into a three-dimensional volume; the resulting volume will be a strong backprojection of $\mathcal{G}$. If subsequent analysis reveals that a larger volume is required, contour expansion may be continued, producing a larger strong backprojection. This expand/lift process may be repeated until either (*i*) a suitable backprojection is found, (*ii*) contour expansion is no longer possible, or (*iii*) the calling program decides to give up and try something else. This control structure is illustrated in Figure 160.

This simple algorithm reduces the problem of constructing strong backprojections to three subproblems: constructing an initial contour surrounding the stable subset of $\mathcal{G}$, expanding a contour to include configurations that will enter the contour, and converting the enclosed surface patch into a three-dimensional volume. The following sections address each of these subproblems, and conclude with a discussion of how the BP_i algorithm may be implemented to support incremental construction of obstacle surfaces.

BP$_i(\mathcal{P}_m, \mathcal{P}_f, D(x, y, \theta, \text{feature}), \mathcal{G}, \mathcal{C}, \textit{stop}?(\text{contour}))$

Construct the configuration-obstacle.
Construct the configuration-space obstacle $\mathcal{O}$ of the polygonal objects $\mathcal{P}_m$ and $\mathcal{P}_f$, using the *CO* algorithm. Assure that $\mathcal{G}$ and $\mathcal{C}$ are disjoint by removing configurations from $\mathcal{G}$ that are also in $\mathcal{C}$.

Construct initial contours.
Call the procedure *construct-initial-contours*$(\mathcal{O}, D, \mathcal{G})$ to construct a set $\mathcal{B}$ of initial contours surrounding the stable subset of $\mathcal{G}$. If $\mathcal{B} = \emptyset$, then return.

Construct the strong backprojection.

$$\left.\begin{array}{c} V_{\text{state}} \\ \mathcal{B}' \\ \text{maximal?} \end{array}\right\} \leftarrow \textit{expand-lift}(\mathcal{B}, D, \mathcal{G}, \mathcal{C}, \textit{stop}?)$$

Return the values V_{state}, $\mathcal{B}'$, maximal?
Return the expanded contours $\mathcal{B}'$ as well as the backprojection volume V_{state} so that additional contour-expansion may be resumed later. The maximal? flag indicates whether additional expansion will be productive; if maximal? is true, then all contours in $\mathcal{B}'$ have been expanded as far as possible. Otherwise, further expansion may be accomplished by additional calls to the *expand-lift* procedure.

expand-lift $(\mathcal{B}, D, \mathcal{G}, \mathcal{C}, \textit{stop}?)$

Expand the input contours.

$$\left.\begin{array}{c} \mathcal{B}' \\ \text{maximal?}' \end{array}\right\} \leftarrow \textit{expand}(\mathcal{B}, D, \mathcal{G}, \mathcal{C}, \textit{stop}?)$$

Construct the strong backprojection volume.

$$\left.\begin{array}{c} V_{\text{state}} \\ \mathcal{B}'' \\ \text{maximal?}'' \end{array}\right\} \leftarrow \textit{lift}(\mathcal{B}', D, \mathcal{G}, \mathcal{C}, \textit{stop}?, \text{maximal?}')$$

Return the values V_{state}, $\mathcal{B}''$, maximal?$''$

Figure 160: The BP_i algorithm.

Creating an Initial Contour

Given a goal $\mathcal{G}$, the algorithm constructs an initial contour surrounding the stable subset of $\mathcal{G}$. This is accomplished by discarding the configurations in $\mathcal{G}$ where lost-contact is possible, constructing contours surrounding the remaining portions of $\mathcal{G}$, and shrinking these contours to remove configurations whose possible motions may exit $\mathcal{G}$. This procedure is illustrated in Figure 161.

This procedure identifies a set of contours describing stable goals. These contours are stable in the sense that once the physical system reaches a configuration in the contour, it will remain in the contour indefinitely. This does not imply that the system will come to rest at any particular configuration; in fact, it may continuously undergo small motions that remain inside the contour. In this sense the contours identify a configuration region that is reliably stable, even though individual configurations within the region may not be reliably stable.

If the physical system satisfies the half-space invariant, the BP_e algorithm may be used to construct an energy-based strong backprojection, which is then used as a backprojection base for the BP_i algorithm. In this case the initial contour is formed around the border of the puddle returned by the BP_e algorithm, and the stability analysis of the *construct-initial-contours* procedure is not required.

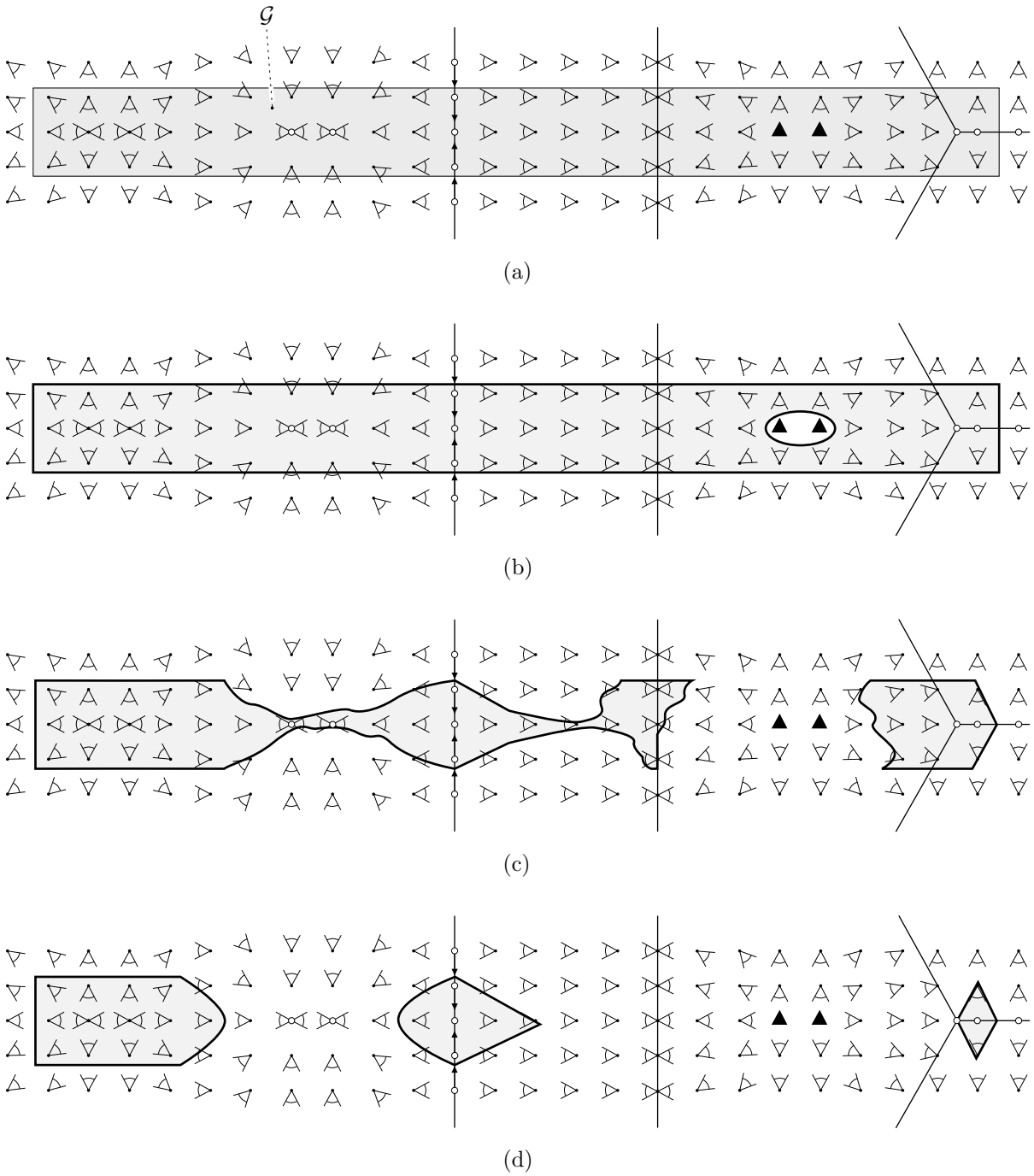


Figure 161: Constructing initial contours. (a) A strange physical system, illustrated in a schematic configuration space. The state-transition field of the physical system is shown; configurations where "no motion" is possible are denoted by open circles at the base of the state-transition cone. Dark triangles indicate configurations where lost-contact is possible. The input goal $\mathcal{G}$ is very broad, allowing all configurations within the shaded rectangle. (b) After discarding configurations where lost-contact is possible, and forming initial contours surrounding the remaining configurations. (c) After several shrinking steps to remove configurations that may exit the contour. (d) After all shrinking steps are complete; these contours delineate stable goal regions.

construct-initial-contours $(\mathcal{O}, D, \mathcal{G})$

This procedure constructs a set of contours surrounding the stable subset of $\mathcal{G}$. The procedure identifies this subset using the function D, which describes the set of possible motions of the moving-object. When the physical system satisfies the necessary assumptions, these contours may also be constructed using the BP_e algorithm; this method is omitted for clarity.

Remove lost-contact goal configurations.
Discard the configurations in $\mathcal{G}$ where lost-contact is possible, forming the set of configurations $\mathcal{G}'$ where contact must be maintained. Accomplish this by calling the function $D(x_{\text{free}}, y_{\text{free}}, \theta_{\text{free}}, nil)$, to obtain the free-space translation directions $[\eta_{\min}, \eta_{\max}]$, and then using these directions and the c-obstacle data structure to directly construct the lost-contact configurations in $\mathcal{G}$.

Construct initial contours.
Construct a contour surrounding each connected component in $\mathcal{G}'$, producing a set of contours $\mathcal{B}$.

Shrink contours to remove unstable configurations.
For each contour $B \in \mathcal{B}$, shrink B to remove configurations that may possibly exit B. This is similar to the contour-expansion process, but requires special code to properly handle cases where a contour crosses itself, crosses another contour, or vanishes during shrinking.

Return the remaining contours.
Each contour remaining in $\mathcal{B}$ encloses a set of configurations $G \subseteq \mathcal{G}$, where no configuration in G has a state-transition-cone that exits G. This implies that once the physical system reaches a configuration in G, it will never leave G. Therefore each contour in $\mathcal{B}$ encloses a set of stable goal configurations; return $\mathcal{B}$.

Figure 161 (continued): The *construct-initial-contours* procedure.

Expanding a Contour

The initial contour constructed by the algorithm is expanded in a greedy fashion until one of the above termination criteria is met. The contour is expanded to include all configurations that will definitely enter the contour, as long as those configurations are not members of the constraint set $\mathcal{C}$. The expansion is performed by applying repeated expansion steps to the contour.

Each of these expansion steps is actually a two-phase process: In the first phase, the algorithm visits every contour-point in the contour, contructs a new expanded position for the contour-point, and moves it there. This phase includes the topological management operations illustrated in Figures 155 through 157. In the second phase, the algorithm again visits every point in the contour, and applies splitting and merging operations as needed to assure that each contour-segment is within the appropriate length bounds. This control structure is shown in Figure 162.

This two-phase expansion process reduces the contour-expansion problem to the problem of constructing a new expanded position for a given contour point P. This is accomplished by constructing the state-transition-cone at P, and constructing a point P' that would cause the bounding rays of the state-transition-cone to fall just inside the adjacent contour-segments. This construction is illustrated in Figure 163. This process is tempered by a maximum step distances $d_{\max}$ and $\delta_{\max}$, which prevent the algorithm from taking large expansion steps when the state-transition cone is very narrow (Figure 164).

Constructing a valid expansion point involves additional considerations: The contour should not be expanded to include P' if lost-contact or static-equilibrium are possible at P', if P' is in the constraint set $\mathcal{C}$, or if expanding the contour to P' causes the contour to cross obstacle features without proper topological representation. The algorithm responds to these situations by reducing the distance between P and P' in an attempt to identify an acceptable expansion point. These and other condsiderations are addressed by the *expand-contour-point* procedure, shown in Figure 165.

expand $(\mathcal{B}, D, \mathcal{G}, \mathcal{C}, stop?)$

Expand each contour.
For each contour $B \in \mathcal{B}$:

Repeat:

changed? $\leftarrow$ false.

Expand the points in the contour.
Call the procedure *expand-contour-point*(contour-point, $D, \mathcal{G}, \mathcal{C}$, changed?) for each contour-point in B. This procedure has the side-effect of setting changed? to true if the contour-point is modified.

Assure that spacing is proper.
Visit each contour-point in B, and split or merge the adjacent segments to bring them within the length bounds $[l_{\min}, l_{\max}]$ and $[\lambda_{\min}, \lambda_{\max}]$. Allow zero-length edges for the appropriate topological situations.

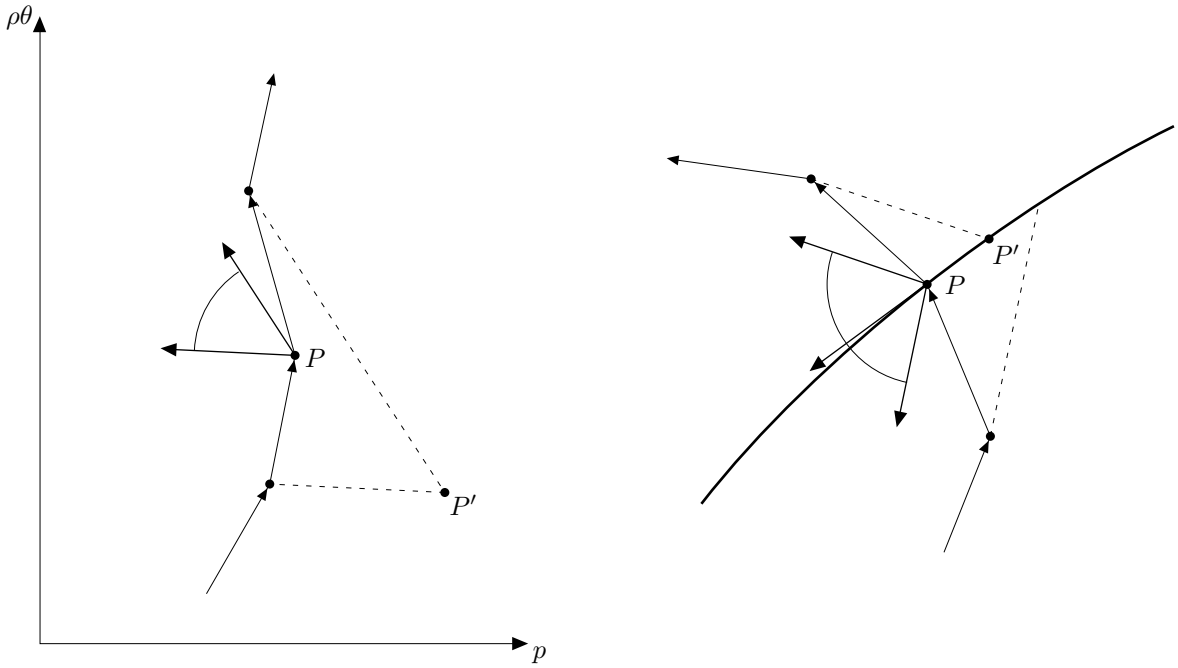
until $stop?(B) = true$ or changed? = false.
The input predicate *stop?* may be an arbitrary decision procedure. This procedure may return true if B reaches a certain size, or ignore B and return true based on the number of iterations or elapsed time. The changed? flag indicates whether this iteration modified the contour; if not, then further iterations are pointless.

endfor

Determine whether further expansion is possible.
Set maximal? to true if all contours in $\mathcal{B}$ have been expanded as far as possible. This may be determined from the changed? flags developed in the previous loops.

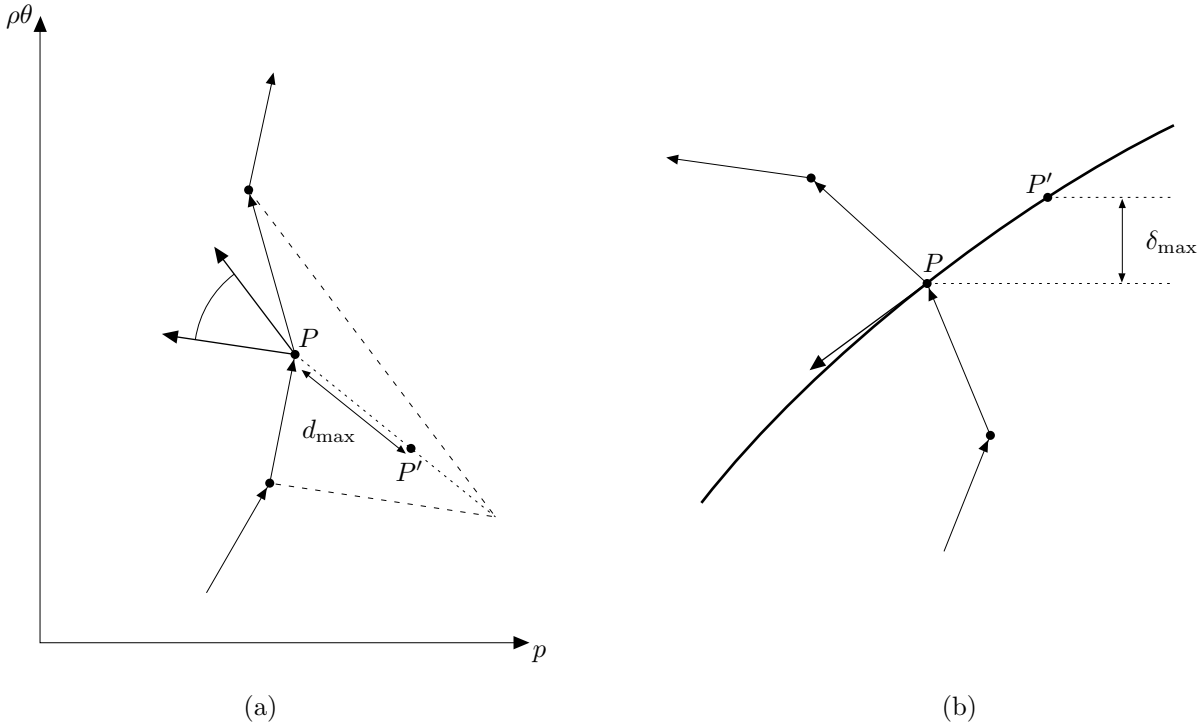
Return the values $\mathcal{B}$, maximal?

Figure 162: The *expand* procedure.



(a) (b)

Figure 163: Constructing an expansion position P' for a contour-point P. (a) If P is on an obstacle facet, the bounding rays of the state-transition cone are combined with the previous and next contour-points to construct P'. This computation is performed in the $(p, \rho\theta)$ space to improve numerical robustness. (b) If P is on an obstacle edge, then the state-transition cone is again used to construct P', but with the additional constraint that P' must remain on the edge. Each state-transition cone ray is analyzed in the $(p, \rho\theta)$ space of the corresponding facet. The edge/facet interface contour-points have been omitted for clarity.



(a) (b)

Figure 164: Limiting the expansion of a contour-point. (a) If the nominal expansion point is too far from P when expanding a facet contour point, an intermediate point is constructed along the line connecting P and the nominal expansion point. (b) A similar limit is applied to the expansion of contour-points on obstacle edges. Here the state-transition cone indicates that pure motion along the boundary will occur, so P' is constructed a distance $\delta_{\max}$ away from P in the appropriate direction. The linear bound $d_{\max}$ is also checked, to prevent large expansion steps on steeply-sloped edges.

expand-contour-point (contour-point, $D, \mathcal{G}, \mathcal{C}$, changed?)

Expand only valid backprojection points.
The recursive procedure that lifts contours into three-dimensional volumes sometimes creates contours that contain invalid backprojection points. These contours should be left intact, because the measure-zero sets they enclose are used to terminate the recursion. Invalid points may be detected by calling the procedure *valid-backprojection-point?*(P, feature, $D, \mathcal{G}, \mathcal{C}$), where the P and feature parameters are obtained from the contour-point. If this procedure returns false, then return.

Expand the contour-point.
Execute one of the following, based on the type field of the contour-point:

`vertex`: Topologically move the contour onto the obstacle features adjacent to the vertex. This is a purely topological modification; all generated contour-points have the same (x, y, θ) field as the input contour-point. Set changed? to true, and return.

`interface`: Depending on the adjacent contour-segments and the edge and facet state-transition cones, it may be appropriate to convert this contour-point to a facet or edge contour-point, or an interface contour-point on the opposite side of the edge. If so, perform the appropriate topological modification, set changed? to true, and return.

`edge`, `facet`: **Construct the expansion point.**
Call the function $D(x_P, y_P, \theta_P, \text{feature})$ to construct the state-transtion-cone for the contour-point. Using the techniques described in the text, construct the point P' such that moving the contour-point to P' would cause the adjacent contour-segments to nearly parallel the bounding rays of the state-transition cone. Special code must be included to handle cases where the adjacent contour-segments are of zero length, which may occur after certain topological modifications. If P' cannot be constructed because of the geometry of the adjacent contour-segments, then return.

Adjust the expansion point.
If the distance between P and P' exceeds $d_{\max}$ in (x, y) or $\delta_{\max}$ in θ, then construct a new P' along the line connecting P and P' that is within these limits.

If P' or the new adjacent segments cross a topological feature on the obstacle surface, then contruct a new P' along the line connecting P and P' that causes the contour to just touch the crossed feature.

Call the procedure *valid-expansion-point?*(P', contour-point, $D, \mathcal{G}, \mathcal{C}$); if this procedure returns false, construct a new P' halfway between P and P', and call the procedure again. If the procedure returns false a second time, then return.

Reposition the contour-point.
Set the (x, y, θ) coordinates of the contour-point to P', and perform the appropriate topological modifications. Set changed? to true, and return.

Return.

Figure 165: The *expand-contour-point* procedure.

valid-expansion-point? $(P', \text{contour-point}, D, \mathcal{G}, \mathcal{C})$

Construct the state-transition cone for P' by calling the function $D(x_{P'}, y_{P'}, \theta_{P'}, \text{feature})$, where the feature parameter is obtained from the contour-point.

If the state-transition-cone contains possible free-motions, then return false.

Else compare the possible motion directions represented by the state-transition-cone with the adjacent contour-segments that would result from moving the contour-point to P'; the details of this test will depend on the type of the contour-point. If there are motion directions that lie outside the new adjacent contour-segments, then return false.

Else call the function *valid-backprojection-point?*$(P', \text{feature}, D, \mathcal{G}, \mathcal{C})$, and return the result.

valid-backprojection-point? $(P, \text{feature}, D, \mathcal{G}, \mathcal{C})$

This routine returns true unless $P \in (\mathcal{Q} - \mathcal{G}) \cup \mathcal{C}$, where $\mathcal{Q}$ is the set of configurations where static equilibrium is possible.

If $P \in \mathcal{G}$, then return true.

Else construct the state-transition cone for P by calling the function $D(x_P, y_P, \theta_P, \text{feature})$; if the no-motion? field of the resulting state-transition-cone is true, then return false.

Else if $P \in \mathcal{C}$, then return false.

Otherwise, return true.

Figure 165 (continued): Predicates called by the *expand-contour-point* procedure.

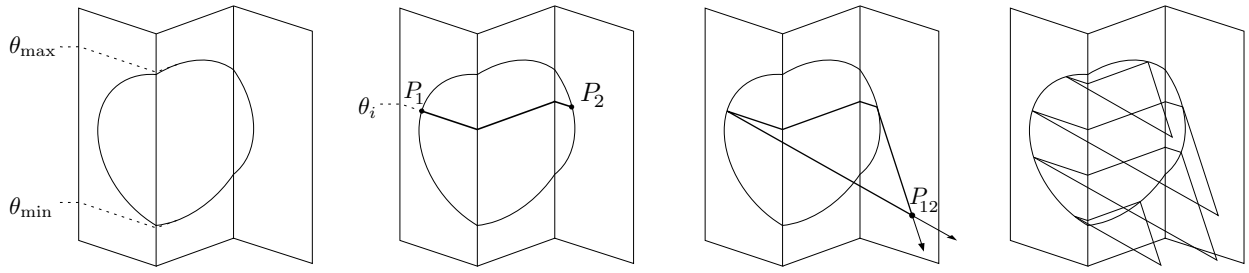


Figure 166: Converting a contour into a volume. (a) A contour on a schematic configuration obstacle, after identifying the bounding θ-values. (b) For a typical slice θ_i, the contour crossing points P_1 and P_2 are found, and these points are connected by a sequence of line segments constructed on the obstacle surface. (c) Rays are erected from P_1 and P_2, opposite the free-space motion directions $\eta_{\min}$ and $\eta_{\max}$. These rays are intersected to form a closed polygon, representing the volume's intersection with a horizontal plane at θ_i. (d) Applying this procedure to a series of θ-values produces a slice-based representation of the backprojection volume.

Converting a Contour to a Volume

After each expansion step, the contour encloses a connected subset of the configuration-obstacle surface. This surface may be lifted into the surrounding free space to form a volume of configurations V_{state}, which is a strong backprojection of $\mathcal{G}$ under the applied action. This lifting operation is performed by a ray-construction procedure similar to Erdmann's algorithm for constructing strong backprojections [Erdmann 1986].

The procedure begins by scanning the contour to find its minimum and maximum θ-values; these bounds are used to construct a set of evenly-spaced θ-values that define a set of constant-θ planes intersecting the contour. Then for each of these θ-values, the algorithm constructs a slice of the strong backprojection volume by finding the points where the contour crosses θ, generating a sequence of line-segments on the obstacle surface connecting these points, and using the free-space translation directions of the state-transition cone to erect rays that convert the line segments into a closed polygon. This construction is illustrated in Figure 166.

The volume constructed by this procedure may re-intersect the configuration obstacle surface; these intersections must be detected and trimmed appropriately to remove unreachable configurations. Further, some of these intersections may identify features where additional backprojection will substantially increase the total backprojection volume, as shown in Figure 167. The algorithm detects these situations and recursively performs additional contour expansion to explore these features. This recursive control structure is reflected in the *lift* procedure, shown in Figures 168 through 170.

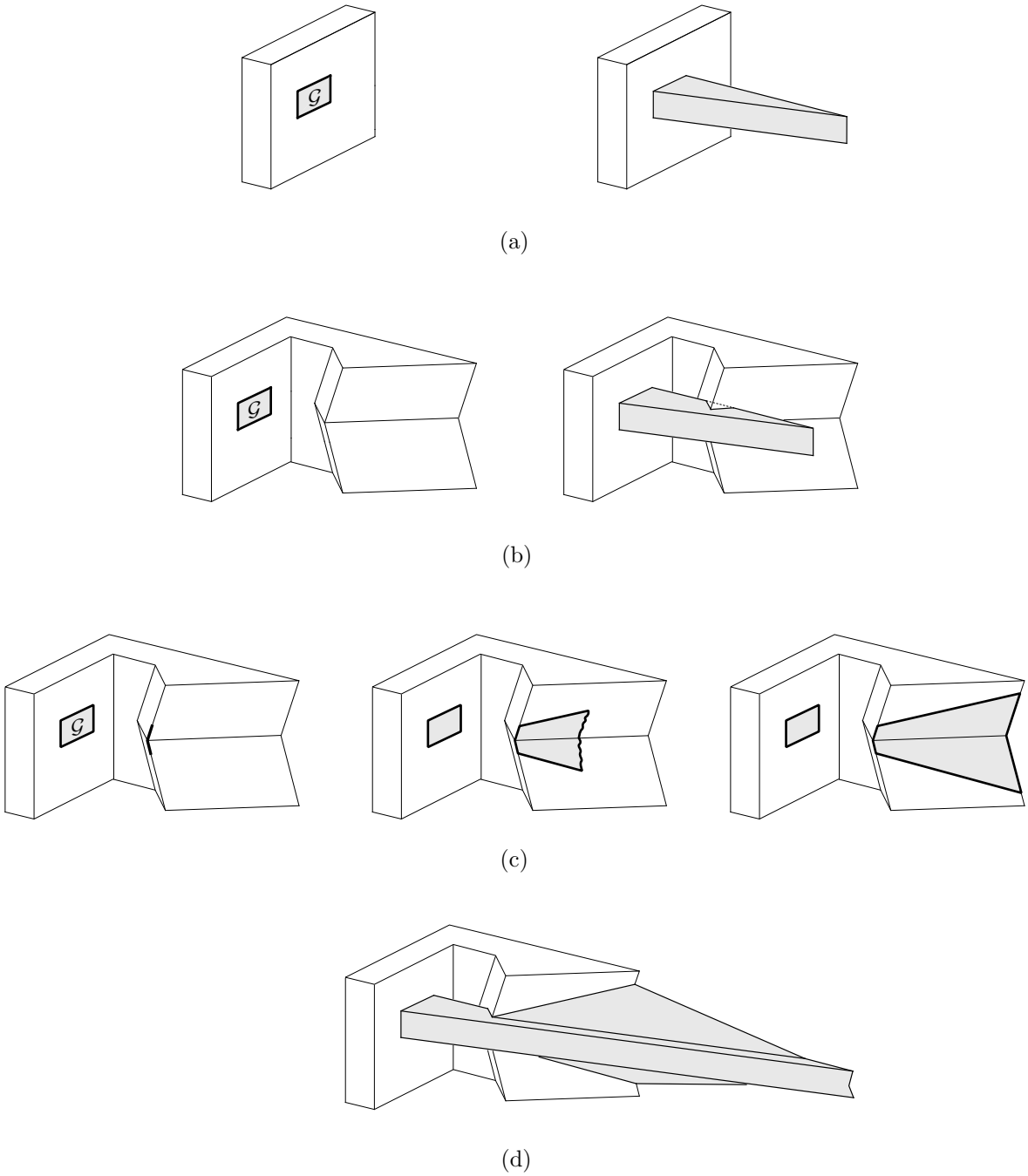


Figure 167: (a) A backprojection of a goal in a schematic configuration space. (b) A backprojection of the same goal, but this time the lifted volume re-intersects the configuration obstacle. (c) The bold edges correspond to the obstacle feature where ray-construction may resume outside the configuration obstacle. A contour is constructed around this set of points, and expanded using the same procedure that expanded the original contour surrounding $\mathcal{G}$. After the new contour is fully expanded, the ray-construction procedure is applied again. (d) The final backprojection volume. Because additional contour-expansion was performed, this volume is much larger than the volume shown in (a).

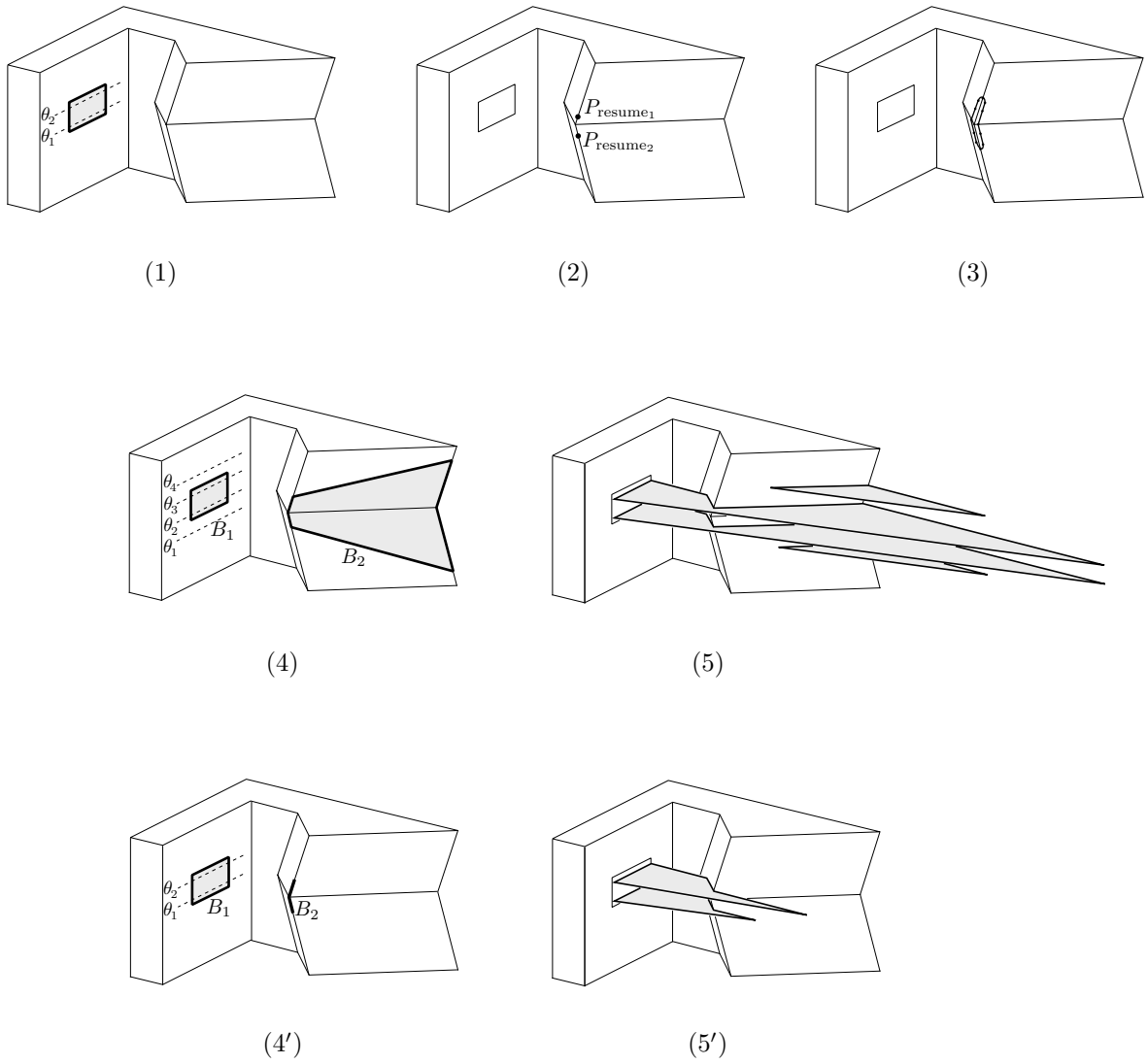


Figure 168: An example execution of the *lift* procedure. (1) Slice θ-values are identified; a coarse θ-resolution will be used in this example for clarity. (2) The *lift-slice* procedure is called, which determines that additional contour-expansion is required. This is indicated by the P_{resume} points returned in the set $\mathcal{R}$. (3) The *lift* procedure constructs "shadow-lines" containing the P_{resume} points, and constructs a contour surrounding the resulting edge-sequence. The contour is then expanded, after which the *lift* procedure recursively calls itself. (4) The recursively-called *lift* procedure now receives two input contours, which span a broader range of θ-values. (5) The *lift-slice* procedure is called again, this time constructing a valid slice on every call. (4′) An alternative outcome. If the surface friction is very high, the contour constructed in (3) does not grow during contour-expansion. The recursively-called *lift* procedure receives two input contours, but in this case the θ-range remains unchanged. (5′) The *lift-slice* procedure constructs valid slices, because of the measure-zero contour B_2.

lift $(\mathcal{B}, D, \mathcal{G}, \mathcal{C}, \mathit{stop}?, \text{maximal}?)$

Identify slice θ-values.
Search the input contours in $\mathcal{B}$ for the minimum and maximum θ-values $[\theta_{\min}, \theta_{\max}]$, and divide this interval into a set $\Theta = \{\theta_1, \theta_2, ..., \theta_n\}$ of evenly-spaced θ-values. Special code should be included to properly handle cases where $\mathcal{B}$ spans all angles.

Initialize V and $\mathcal{R}$.
Construct a c-volume data record and set its θ-interval field to $[\theta_{\min}, \theta_{\max}]$. Set $\mathcal{R}$ to the null set.

Construct each slice.
For each $\theta \in \Theta$:

Call the procedure *lift-slice*$(\theta, \mathcal{B}, D, V, \mathcal{R})$, which uses a ray-construction procedure to construct a volume slice at θ, or identify points requiring further contour-expansion. If further contour-expansion is not required, the constructed slice is placed in the volume V; otherwise, the points requiring contour-expansion are placed in the set $\mathcal{R}$.

endfor

Decide whether further contour-expansion is required.
If $\mathcal{R} = \emptyset$, then

Contour expansion is not required.
V now contains a collection of valid slices. Return the values V, $\mathcal{B}$, maximal?

else

Contour expansion is required.

Construct "shadow lines" on the obstacle.
Construct a sequence of obstacle edges for every portion of the obstacle that occluded the ray-construction procedure. This is accomplished by calling the procedure *construct-shadow-lines*$(\mathcal{R})$, which returns a set $\mathcal{E}$ of edge-sequences corresponding to shadow lines.

Convert the shadow lines into expandable contours.
Construct a contour B around each edge-sequence in $\mathcal{E}$, producing a set of contours $\mathcal{B}_{\text{new}}$. Do not apply the stability test used to filter the initial contours surrounding $\mathcal{G}$; all contours in $\mathcal{B}_{\text{new}}$ must be retained to prevent infinite recursion.

Expand the new contours.
$$\left.\begin{array}{c}\mathcal{B}'_{\text{new}} \\ \text{maximal}?_2\end{array}\right\} \leftarrow \mathit{expand}(\mathcal{B}_{\text{new}}, D, \mathcal{G}, \mathcal{C}, \mathit{stop}?)$$

Merge the new and original contours.
$\mathcal{B}_{\text{all}} \leftarrow \mathcal{B} \cup \mathcal{B}'_{\text{new}}$
$\text{maximal}?_{\text{all}} \leftarrow \text{maximal}? \wedge \text{maximal}?_2$

Recurse.
$$\left.\begin{array}{c}V \\ \mathcal{B}'_{\text{all}} \\ \text{maximal}?'_{\text{all}}\end{array}\right\} \leftarrow \mathit{lift}(\mathcal{B}_{\text{all}}, D, \mathcal{G}, \mathcal{C}, \mathit{stop}?, \text{maximal}?_{\text{all}})$$

When this procedure recurs, all of the P_{resume} points that required contour expansion will produce segments in the *lift-slice* procedure, although some segments may be of zero length. Thus these points will not require further contour-expansion and recursion will terminate, unless new P_{resume} points are identified. Such points can only arise a finite number of times, so recursion will eventually terminate.

Return the values V, $\mathcal{B}'_{\text{all}}$, $\text{maximal}?'_{\text{all}}$

Figure 168 (continued): The *lift* procedure.

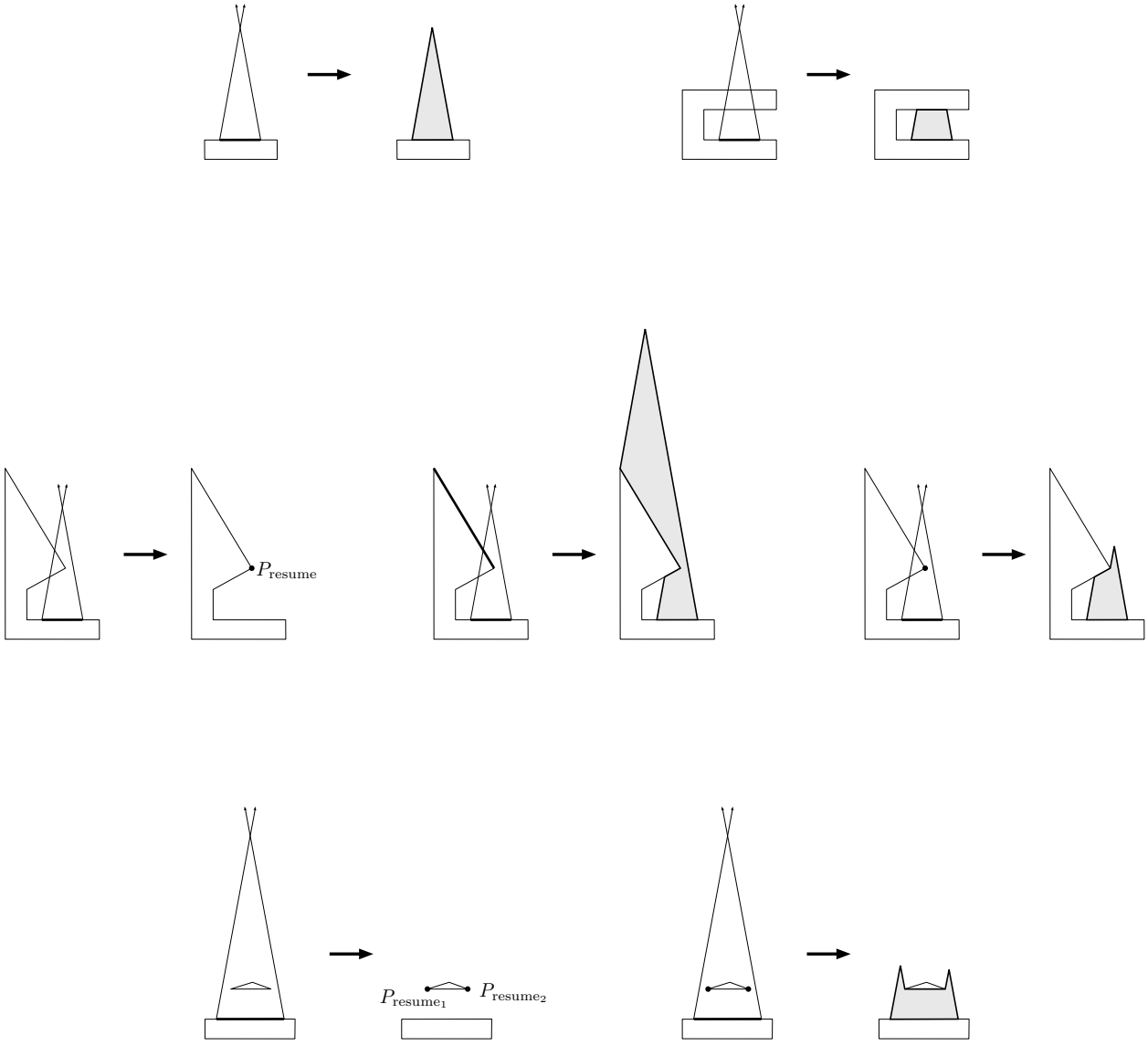


Figure 169: Example executions of the *lift-slice* procedure. The middle row corresponds to the example shown in Figure 168.

lift-slice $(\theta, \mathcal{B}, D, V, \mathcal{R})$

Construct enclosed obstacle segments.
Search each contour in $\mathcal{B}$ for points where the contour crosses the input θ-value, and connect these points with line segments that represent the portion of the obstacle surface enclosed in $\mathcal{B}$. Include zero-length segments in this construction. Attach adjacent segments to form segment-strings, and associate each string with the contour that produced it. Place these strings in the set $\mathcal{S}$.

Apply ray-construction.
For each segment string $S \in \mathcal{S}$:

If S is already part of a polygon in $\mathcal{P}$, then ignore S.

Otherwise, erect rays from the endpoints of S, forming a closed polygon. Check for points where this polygon intersects the obstacle surface, including enclosed polygonal regions. If no intersections are found, add the polygon to $\mathcal{P}$.

If an intersection is found, attempt to resolve the intersection by identifying the points P_{resume} where ray-construction may proceed, and connect these points to each other or the polygon boundary appropriately. Attempt to match the P_{resume} points to the endpoints of other strings in $\mathcal{S}$; if this succeeds, then connect the strings and attempt ray-construction again. Otherwise, construct a data record $\langle P_{\text{resume}_i} \ \mathcal{B}_i \ \eta_i \rangle$ describing each unmatched P_{resume} point, where $\mathcal{B}_i$ is the set of contours contributing to S, and η_i is the direction giving rise to P_{resume_i}. Place these records in $\mathcal{R}$, and return.

endfor

Update the backprojection volume.
At this point, all P_{resume} points must have matched strings in $\mathcal{S}$, and $\mathcal{P}$ must contain a collection of polygons describing a valid slice. Construct a c-slice with θ and the set of polygons $\mathcal{P}$, and add it to the slice-list for V.

Return.

Figure 169 (continued): The *lift-slice* procedure.

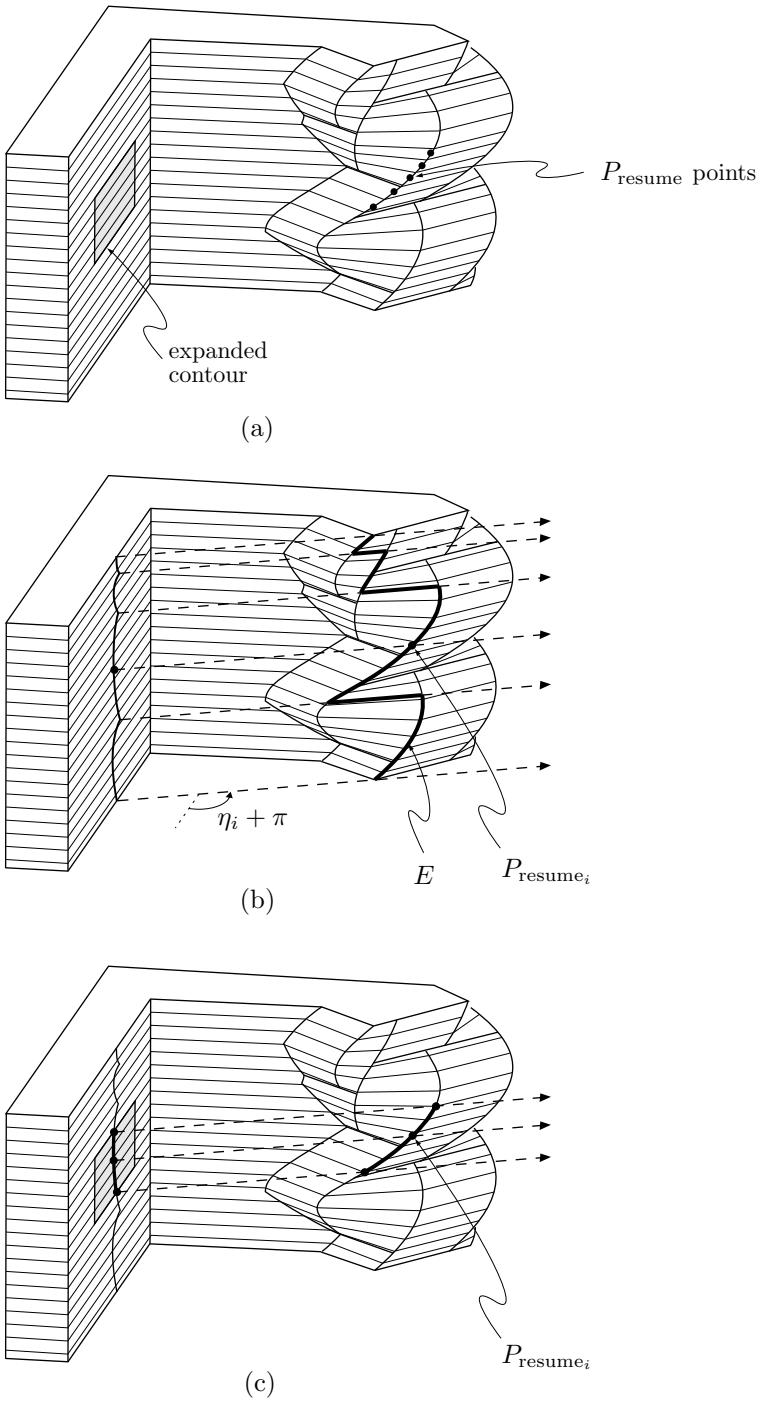


Figure 170: Generating shadow lines on the obstacle surface. (a) The procedure begins with a collection of P_{resume} points, which correspond to shadow points discovered during slice analysis. (b) The unbounded shadow line E is constructed, shown by the bold curve. This sequence of obstacle edges may be generated directly from η_i and P_{resume_i}, using the c-obstacle data structure. The projection of E onto the surface containing the contour is shown for clarity; this curve is not constructed by the procedure. (c) A finite subset of E is identified by an iterative numerical search proceeding along E in both directions from the point P_{resume_i}. At each iteration, the current point on E is projected back onto the surface containing the contour; the bounds of E correspond to the points where this projection leaves the contour.

construct-shadow-lines ($\mathcal{R}$)

This procedure constructs a sequence of obstacle edges for every portion of the obstacle that occluded the ray-construction procedure. These edges correspond to the set of P_{resume} points that would be constructed if the *lift-slice* procedure performed its computation over continuous θ-intervals instead of discrete θ-values. These edges are constructed using the adjacency and metric information of the c-obstacle data structure; the endpoints of the sequence are identified via numerical search. The P_{resume} points identified in the *lift-slice* procedure are used as seeds for the numerical search.

Repeat:

Select and remove a data record $\langle P_{\text{resume}_i} \;\; \mathcal{B}_i \;\; \eta_i \rangle$ from $\mathcal{R}$.

Construct the sequence of critical edges E that contain P_{resume_i} and form a shadow line with respect to the direction η_i.

Starting at the point P_{resume_i}, iteratively trace the edges in E in both directions, projecting a ray in the direction η_i onto the obstacle surface. Note the points where the projected ray passes outside the contours in $\mathcal{B}_i$; these points delimit the relevant portions of the shadow line. Update E to reflect these endpoints, and place the resulting connected edge-sequence in the set $\mathcal{E}$.

Remove all records from $\mathcal{R}$ whose P_{resume} point is contained in the edge-sequence E; this prevents redundant construction of shadow lines.

until $\mathcal{R}$ is empty.

Return the set of edge-sequences $\mathcal{E}$.

Figure 170 (continued): The *construct-shadow-lines* procedure.

The *lift* procedure shown in Figure 168 may omit configurations from the strong backprojection in certain situations; Figure 171 shows an example. Given the contour shown in (a), the lift procedure constructs a contour B_2 surrounding the shadow line, and passes B_2 to the *expand* procedure. No expansion will result, because "lost-contact" is possible for all configurations on the obstacle facet F. This causes the lift procedure to construct the volume shown in (b). However, a larger backprojection is possible (c). This backprojection may be obtained by informing the *expand* procedure that "lost-contact" motions are acceptable for configurations within the shaded region shown in (d), which is constructed by projecting the original contour onto the obstacle surface. The *lift* procedure may be extended to include these cases; the details of this extension are left for future work.

A second observation regarding the *lift* procedure is that it may include points on the boundary of V_{state} that are not valid backprojection points. For example, this may occur when a ray is erected from a P_{resume} point where static equilibrium is possible, or when the back side of the configuration obstacle contains states in $\mathcal{C}$. These conditions invalidate the strong backprojection, but only on the boundary of V_{state}. These semantic problems can be eliminated by interpreting the strong backprojection V_{state} produced by the BP_i algorithm as an open set that does not include its boundary.

Supporting Incremental Obstacle Construction

The BP_i algorithm may be implemented with a control structure that takes advantage of the incremental construction capability of the *CO* algorithm. The description of the goal $\mathcal{G}$ may be used to identify the obstacle features that should be initially constructed, and the adjacency information of the resulting c-obstacle data structure may be used to identify additional required features as contour-expansion proceeds. When the contour crosses an obstacle edge or vertex onto a facet that has not been constructed, the contour-expansion procedure may call the *complete-facet* procedure to construct the facet, and contour-expansion may resume normally.

As with the BP_e algorithm, the procedure that lifts two-dimensional surface patches into three-dimensional volumes must be able to detect situations where non-local obstacle features intersect the constructed volume. When the configuration obstacle is constructed incrementally, the *lift* procedure must be able to detect these situations for obstacle features that are not yet constructed. It may be possible to accomplish this by adding decision predicates to the *lift* procedure analogous to the *conflict-possible?* tests employed by the *CO* algorithm.

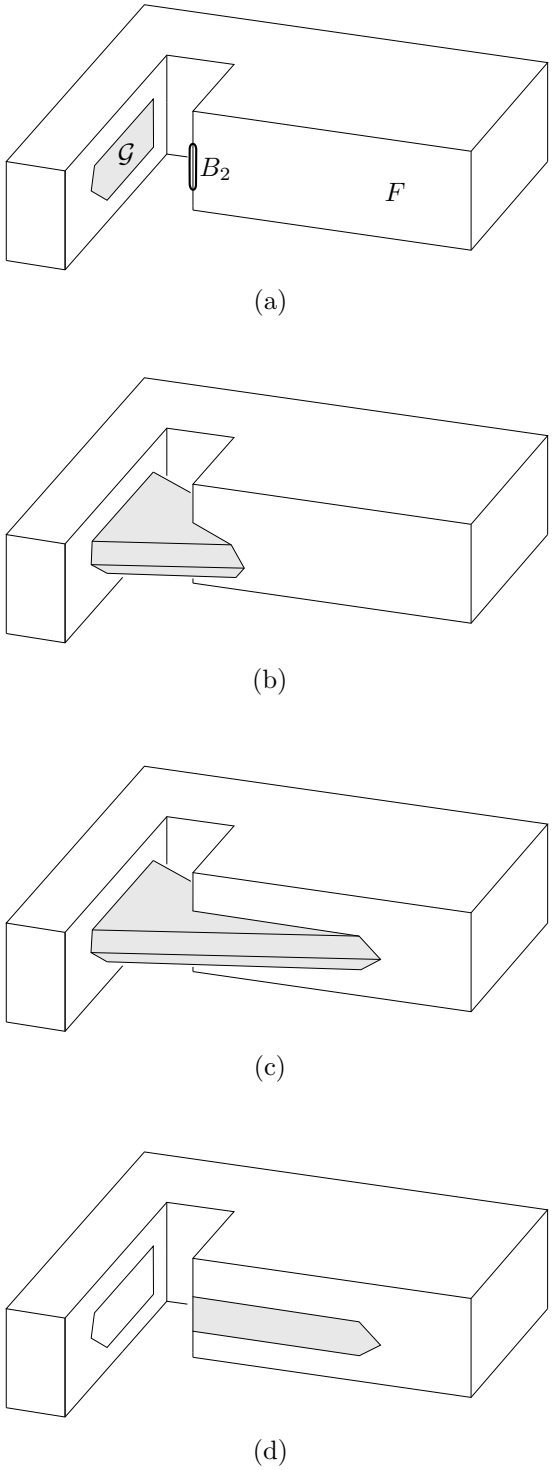


Figure 171: A case where the *lift* procedure omits configurations from the strong backprojection. (a) A goal on a schematic configuration-space obstacle, and the contour B_2 surrounding the shadow line. This contour cannot be expanded, since lost-contact is possible for all configurations on F. (b) The volume constructed by the *lift* procedure. (c) The maximal strong backprojection. (d) The projection of the goal contour onto F; lost-contact is allowable within this region.

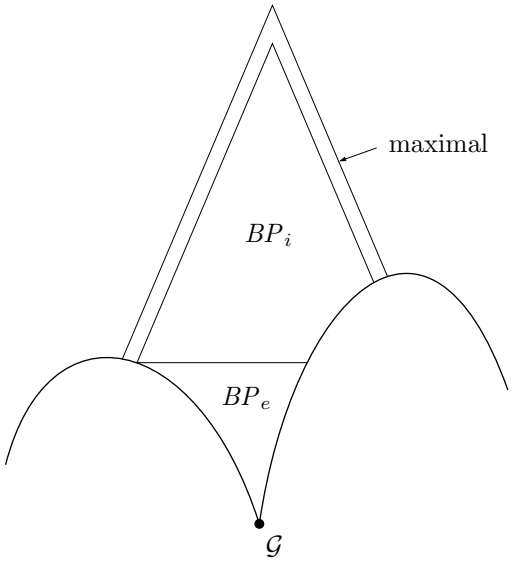


Figure 172: Comparing backprojections. The weaker mechanics model employed by the BP_e algorithm causes it to produce a much smaller backprojection than the BP_i algorithm. If the goal is connected, the BP_i algorithm constructs a backprojection that is slightly smaller than the maximal backprojection, due to discrete approximation errors.

Critique

The BP_i algorithm attempts to identify the largest possible set of initial configurations that will achieve the desired goal. Because the BP_i algorithm utilizes a more detailed model of the physical system behavior, it is capable of constructing backprojections that are much larger than the backprojections produced by the BP_e algorithm. Are these backprojections maximal?

If the stable subset of the goal is connected, then the backprojection V_{state} produced by the BP_i algorithm is nearly-maximal. The difference between the true maximal backprojection and V_{state} is completely an artifact of the discrete approximation employed by the BP_i algorithm. As the granularity of this approximation becomes increasingly fine, V_{state} will approach the strong backprojection, assuming that the anomaly shown in Figure 171 does not arise. Figure 172 shows the relationship between the maximal strong backprojection and the backprojections produced by the BP_e and BP_i algorithms.

If the set of stable goal configurations is disconnected, the BP_i algorithm may produce a backprojection that is significantly smaller than the maximal backprojection, as shown in Figure 173. In this example, a polygonal object is pushed by a fence, and dithering is added to prevent unstable equilibrium configurations. In the configuration space, the goal corresponds to two disjoint sets of stable configurations separated by an obstacle facet (b). Because of uncertainty in the pushing direction, the state-transition cones on the ridge of the facet have an ambiguous rotation direction; this limits the contour-expansion performed by the BP_i algorithm, which produces a disconnected backprojection (c). However, since static equilibrium is not possible along the facet ridge, the true maximal backprojection is a connected set that includes configurations omitted by the BP_i algorithm (d).

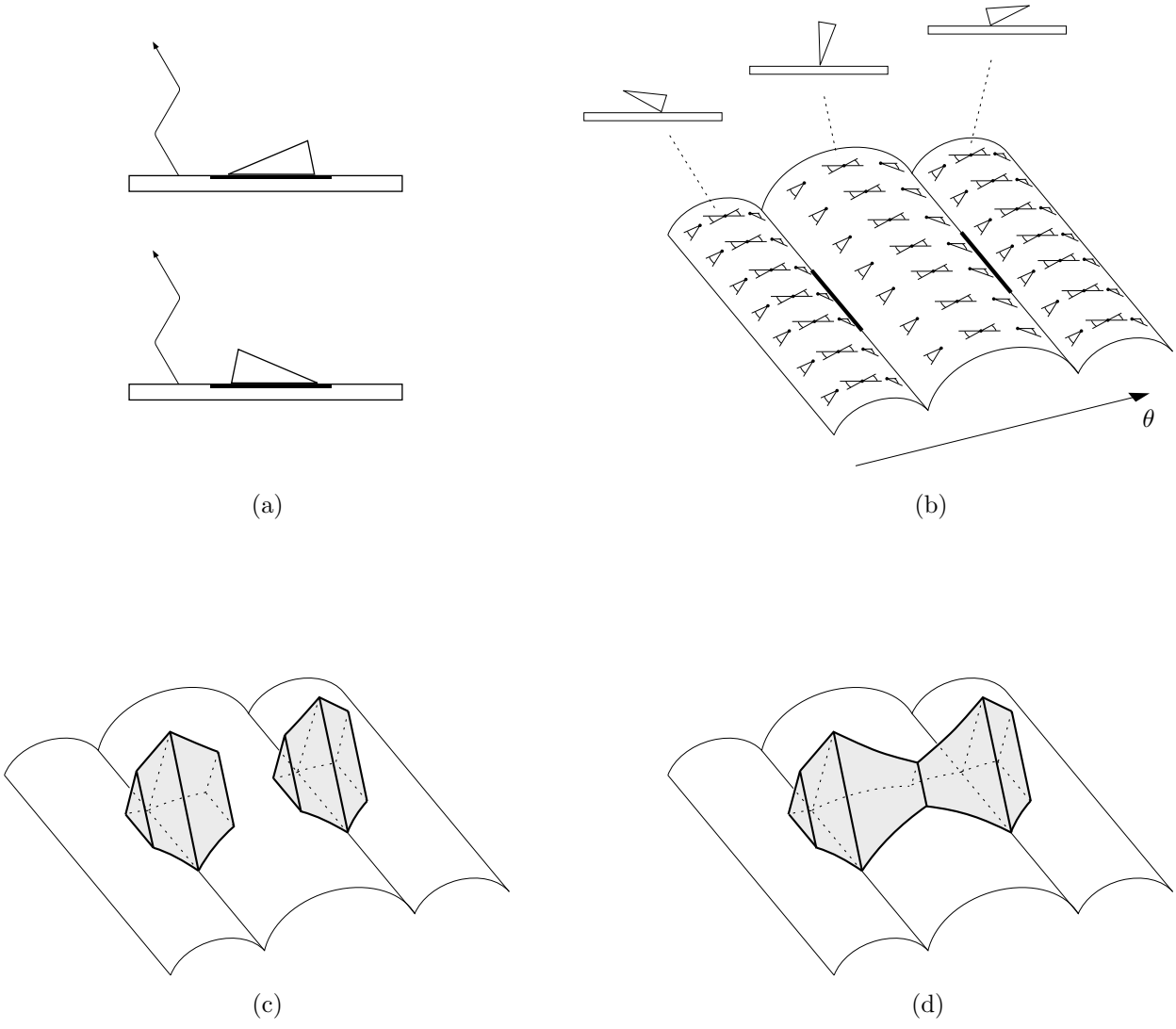


Figure 173: A example where the BP_i algorithm produces a backprojection signficantly smaller than the maximal strong backprojection.

This discrepancy is due to a fundamental limitation of the BP_i algorithm: Since the contour-expansion procedure only includes configurations that will reliably enter a single connected goal set, it is unable to recognize configurations that will not reliably enter any particular connected goal, but must enter one of a collection of connected goals. This is a previously-known limitation of the greedy approach to constructing strong backprojections [Erdmann 1986].

The remedy to this problem is to construct the weak backprojections of all non-goal final configurations, form the union of these weak backprojections, and then complement the resulting set. The BP_i algorithm does not take this approach, because of its computational expense. However, it may be possible to extend the BP_i algorithm to efficiently merge disconnected backprojection volumes, by using the volume boundaries to identify relevant weak backprojections.

Key Assumptions

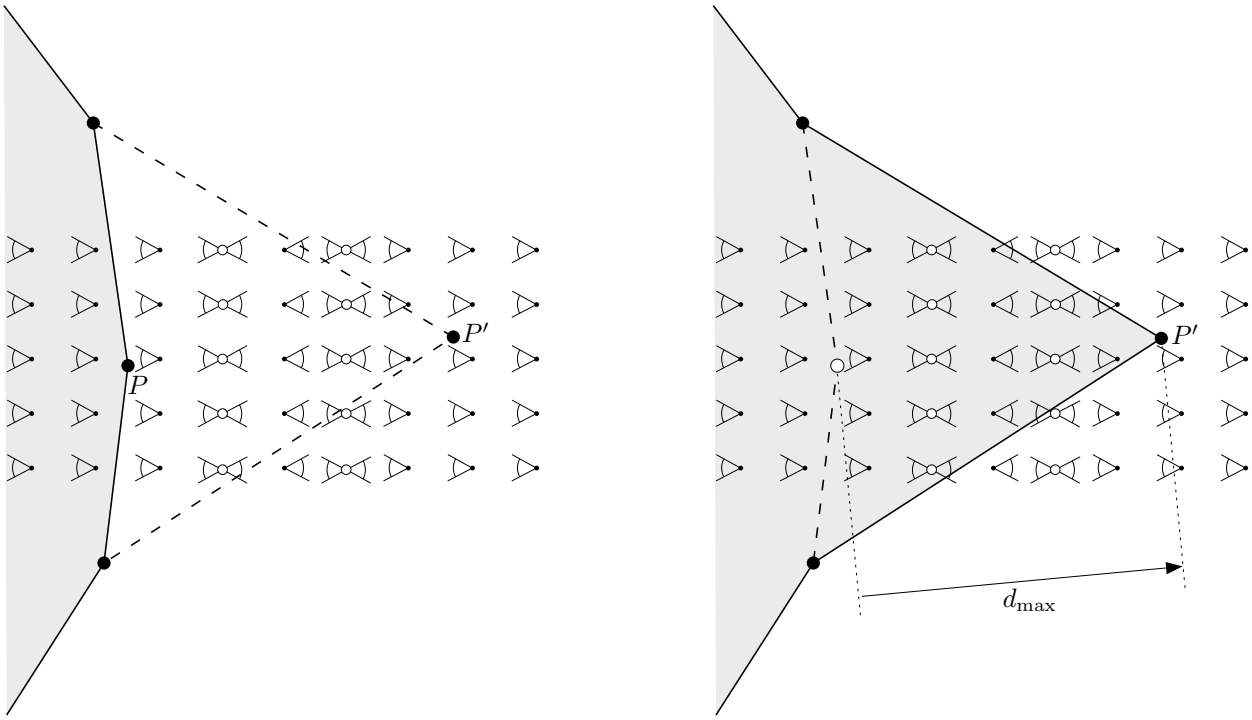
One of the most significant advantages of the BP_i algorithm is its generality: It may be applied to any physical system involving two polygonal objects whose possible differential motions may be bounded by a function $D(x, y, \theta, \text{feature})$, and whose free-space motions are pure translations. Since the D function only addresses the instantaneous behavior of the system, the BP_i algorithm may be applied to physical systems whose trajectories cannot be described by closed-form equations.

The BP_i algorithm is predicated on the assumption that an appropriate D function can be formulated. Can such a function always be found? Trivially the answer is yes, since a D function that always returns "all motions are possible" provides a correct conservative description of all physical systems. However, this function is obviously useless, since it will cause the BP_i algorithm to always return a null backprojection. A useful D function must satisfy a more nebulous criterion: It must be specific enough to allow the BP_i algorithm to produce backprojections large enough to solve the problem at hand. This operational definition clearly depends on the details of each problem instance, and launches us into the perpetual scientific endeavor of developing increasingly precise physical models to solve increasingly difficult analysis problems.

Some physical systems are simpler to analyze than others. For example, generating the D function for a compliant-motion task is often a matter of simply expressing the control law in the appropriate parameter space, and performing a force/motion analysis; see [Erdmann 1984] or [Buckley 1989] for examples. In contrast, linear pushing motions have been studied in at least three Ph.D. theses [Mason 1982; Peshkin 1986; Goyal 1989], but construction of a suitable D function for this work still required a conjecture to predict the motion of the pushed object in multiple-contact situations.

Despite the difficulty of these analysis problems, there may be few alternatives available if we want to build reliable manipulation systems. Uncertainty is inevitably present in real manipulation tasks, and the geometry and mechanics of the task play a dominant role in determining the success or failure of an attempted action. The difficulty we experience in constructing the necessary predictive D functions is in some sense a reflection of the complexity of the task itself.

A second fundamental assumption of the BP_i algorithm is that the physical system behavior varies in a well-behaved manner over the obstacle surface. The BP_i algorithm is designed to properly handle gross changes in the system behavior that occur at topological features on the obstacle surface, but pathological variations within a single feature may pass undetected. Figure 174 shows an example where the BP_i algorithm fails due to high-frequency changes in the physical system's state-transition field.



(a) (b)

Figure 174: A failure in the coutour-expansion procedure caused by high-frequency changes in the physical system behavior. (a) A magnified view of an obstacle facet and its state-transition field. When expanding the contour-point P, the algorithm examines the state-transition cone at P, and constructs the point P'. Since the state-transition cone at P' is contained within the resulting adjacent contour-segments, the contour-point is repositioned to P'. (b) After expanding the contour-point. The contour now encloses non-goal possible-equilibrium configurations, which invalidates the strong backprojection. If $d_{\max}$ had been chosen to be half as large, this error would have been detected.

Because the BP_i algorithm analyzes physical systems whose behavior cannot be described analytically, it is forced to use discrete approximations of continuous sets, and construct these approximations using a finite number of sample points. Consequently the BP_i algorithm includes several "magic numbers" that control its operation; these include the contour-segment length bounds $[l_{\min}, l_{\max}]$ and $[\lambda_{\min}, \lambda_{\max}]$, the maximum expansion-step bounds $d_{\max}$ and $\delta_{\max}$, and the volume slice resolution $\Delta\theta$. The accuracy of the BP_i algorithm will increase as these numbers are made smaller, neglecting finite-precision numerical issues. These parameters should be chosen judiciously, but as of this writing only *ad hoc* techniques are available for selecting these values. It may be possible to use the size of obstacle surface features as a rational basis for choosing these parameters, but errors due to system mechanics (such in Figure 174) will still be possible. Eliminating these errors must be accomplished on a per-system basis, and may be very difficult.

Implementation Issues

I have not implemented the full BP_i algorithm, and so complete robustness and performance data are not available. A partial implementation was used to generate the examples presented in this thesis; no effort was devoted toward improving the robustness or performance of this program.

The practical efficiency of the BP_i algorithm remains to be seen. The current partial implementation is somewhat slow, requiring 38 minutes to construct the example backprojection shown in Figure 10 in Part I. However, several enhancements may dramatically speed up this program; these include implementing expensive components of the pushing D function as table-lookup operations, caching often-referenced constructions in the contour-point data record, and replacing the loop that visits every contour-point with a buffer listing the contour-points that may require expansion. These modifications reduce the number of calls to the *expand-contour-point* procedure, as well as the cost of each call. Similar enhancments may be applied to improve the performance of the *lift-slice* procedure.

Extensions of the BP_i Algorithm

The BP_i algorithm may only be applied to physical systems whose free-space motions are pure translations, in a direction bounded by a constant interval $[\eta_{\min}, \eta_{\max}]$. This limitation is imposed by the ray-construction procedure that lifts the two-dimensional contour into a three-dimensional volume.

We may eliminate this restriction by replacing the two-dimensional contour with a three-dimensional "membrane" that expands into free space as well as along the obstacle surface. By modifying the state-transition-cone to represent possible free-space motions as polygons on the unit sphere, the resulting contour expansion algorithm may be used to construct strong backprojections for physical systems with arbitrary free-space motion characteristics.

Generalizing 2-d contours to 3-d membranes provides other advantages in addition to expanding the class of allowable physical systems. For example, goals that include free-space points may be specified. Further, the *lift* procedure is eliminated, simplifying the overall algorithm. The cost of this additional generality and simplicity is increased combinatorial complexity, since the contour-expansion process fills a three-dimensional volume rather than a two-dimensional area. Further work is required to assess the practical implications of this extension.

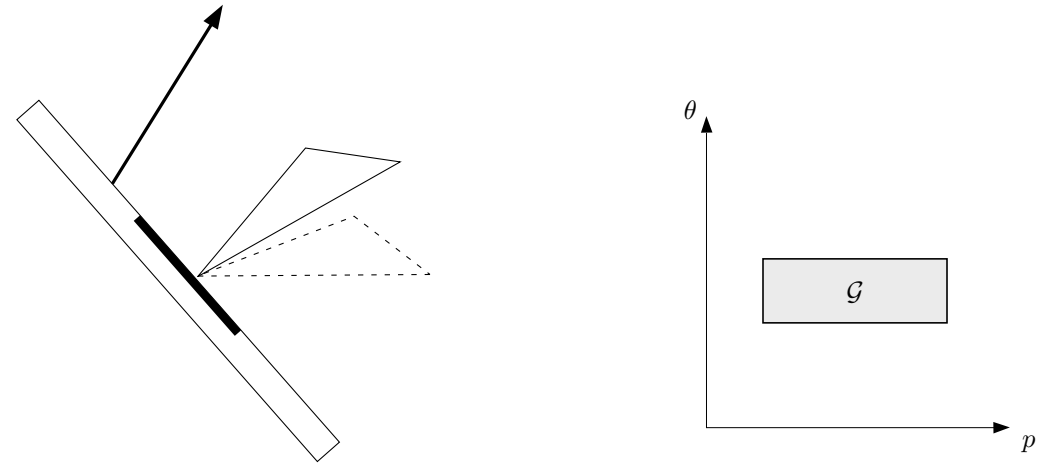
Another key limitation of the BP_i algorithm is its lack of duration information. For example, the algorithm is capable of constructing a set of initial states from which a given pushing action will achieve a desired final configuration, but the algorithm does not identify the pushing distance required to achieve this configuration. This weakens the utility of the strong backprojection that is constructed.

It may be possible to extend the BP_i algorithm to compute duration information. This would require extending the state-transition cone function D to return rate information, and modifying the contour-expansion procedure to maintain a conservative estimate of the integral of this rate information. Such an extended version of the BP_i algorithm would be able to identify the minimum action duration required to assure that a stable goal is achieved, given an initial configuration within V_{state}.

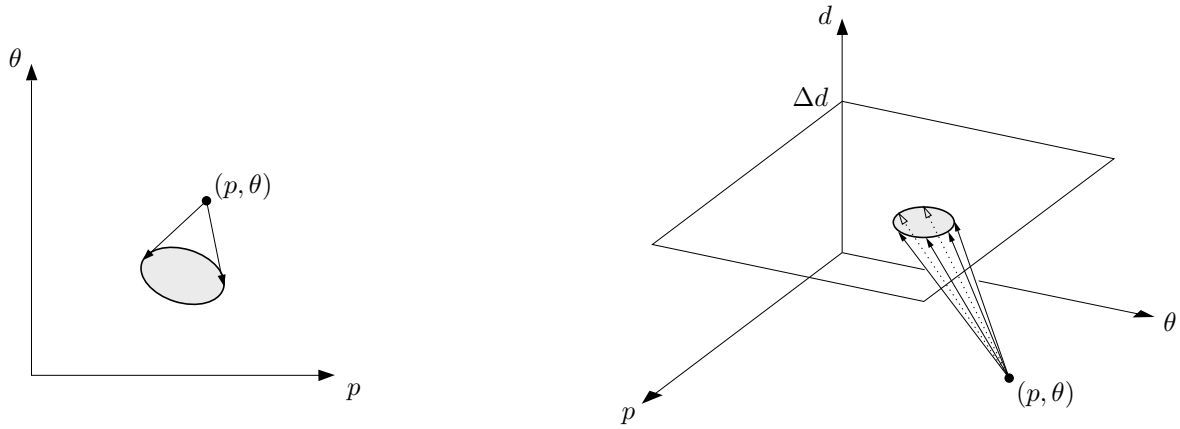
For goals that are not stable, both minimum and maximum duration bounds must be included. This is illustrated in Figure 175, which shows the construction of a strong backprojection for an unstable pushing goal. Given an initial configuration, this backprojection describes the minimum and maximum pushing distances that will reliably achieve the goal. The backprojection is constructed by including pushing distance as an additional dimension in the configuration space, and constructing the backprojection in this extended space. Extending the BP_i algorithm to implement this approach will clearly require substantial effort.

* * *

Analyzing the dynamic behavior of physical systems in the presence of uncertainty is a difficult problem that requires consideration of the geometry, mechanics, and uncertainty that characterize the physical system. The BP_e and BP_i algorithms may be used to solve a restricted class of these problems, but these results represent only a small step toward the general dynamic analysis of the interaction of two planar objects. Enhancing these algorithms to include arbitrary free-space motions, a sense of time, speed information, and detailed collision models are but a few of the possible areas for future work. Many open problems remain.

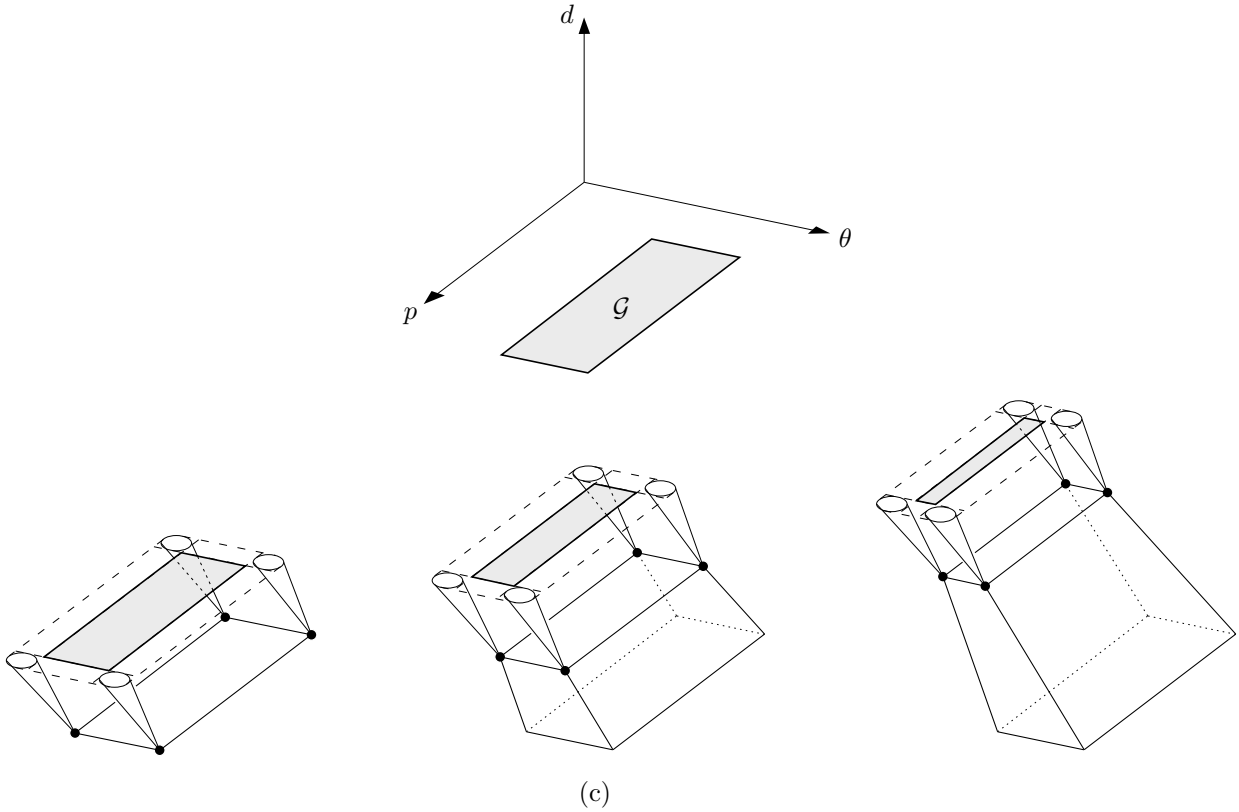


(a)

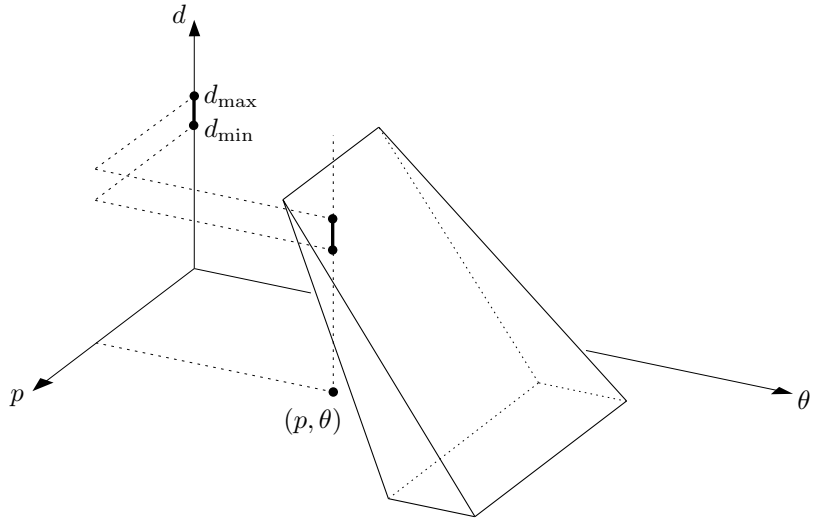


(b)

Figure 175: Constructing a backprojection with a sense of time. (a) An unstable goal configuration. We would like to synthesize a pushing motion such that when the motion terminates, the triangle will contact the highlighted edge segment between the orientations shown. The region $\mathcal{G}$ expresses this goal in the (p, θ) space that parameterizes this ve contact condition. (b) A state-transition cone that contains rate information. If an incremental push of length Δd is executed from an initial configuration (p, θ), the final (p, θ) configuration will lie somewhere in the shaded region. The (p, θ) space may be augmented by adding the pushing distance d as a third dimension; this gives the state-transition cone a three-dimensional interpretation.



(c)



(d)

Figure 175 (continued): (c) Constructing the strong backprojection of the goal in the (p, θ, d) space. Here a state-transition cone is erected at each boundary point of $\mathcal{G}$, and the strict interior of the resulting set is formed. This region is shown shaded, and corresponds to the set of initial configurations that will reliably achieve $\mathcal{G}$ after pushing action of length Δd. This process is repeated, generating the set of initial configurations that will achieve $\mathcal{G}$ for pushing distances of $2\Delta d$, $3\Delta d$, and so on. (d) The final strong backprojection that is constructed. This set is shown with linear boundaries for clarity; the true backprojection will be nonlinear. The point (p, θ) shows an example initial configuration; $d_{\min}$ and $d_{\max}$ correspond to the minimum and maximum pushing distances which will cause this configuration to achieve the goal.

Formulating Motion Commands

The dynamic analysis performed by the BP_e and BP_i algorithms produces a set of configurations V_{state} that correspond to initial states from which a given action will reliably achieve a desired goal. To synthesize a reliable manipulation action from this information, we must identify a commanded initial position which will reliably produce an initial state in V_{state}.

This is accomplished by the *COMMAND* algorithm, which transforms the set of successful initial states V_{state} into a set of successful commanded initial positions V_{command}. V_{command} is constructed so that all commanded positions in V_{command} will reliably produce a true (x, y, θ) configuration in V_{state}, despite the presence of sensing and control errors. This algorithm is illustrated in Figure 176.

The constructions employed by the algorithm are straightforward; the most difficult of these is the volume-shrinking operation, which is easy to perform on volumes represented with slices. Shrinking a slice-based volume by an amount ϵ_{xy} and ϵ_θ is accomplished by shrinking each slice by ϵ_{xy}, and then intersecting each slice with all other slices within a θ-distance of $\pm\epsilon_\theta$. This construction is illustrated in Figure 179; the output of the BP_e algorithm must be converted to a slice-based representation before this construction may be applied.

COMMAND $(V_{\text{state}}, \mathcal{U}_m, \mathcal{U}_f, x, y, \theta, \text{moving?})$

If moving?=true then

$V_{\text{command}} \leftarrow COMMAND_{\text{moving}}(V_{\text{state}}, \mathcal{U}_m, \mathcal{U}_f, x, y, \theta)$

else

$V_{\text{command}} \leftarrow COMMAND_{\text{fixed}}(V_{\text{state}}, \mathcal{U}_m, \mathcal{U}_f, x, y, \theta)$

endif

Return V_{command}.

Figure 176: The *COMMAND* algorithm.

COMMAND$_{\text{moving}}(V_{\text{state}}, \mathcal{U}_m, \mathcal{U}_f, x_f, y_f, \theta_f)$

V_{state} defines a set of true positions of the moving-object origin, measured relative to the true position of the fixed-object origin, and (x_f, y_f, θ_f) is the nominal position of the fixed-object origin, measured relative to the robot's coordinate frame. We seek to transform V_{state} into a volume V_{command}, defining a set of points measured in the robot's coordinate frame, such that if the robot commands a motion of the moving-object to a position in V_{command}, the resulting true position of the moving-object origin measured relative to the fixed-object origin is in V_{state}.

Compensate for $\mathcal{U}_f$.
Form the volume V_1 by applying the transform

$$\begin{aligned} x_1 &\leftarrow -(x_{\text{state}} - x_{C_f})\cos(\theta_{\text{state}}) - (y_{\text{state}} - y_{C_f})\sin(\theta_{\text{state}}) \\ y_1 &\leftarrow (x_{\text{state}} - x_{C_f})\sin(\theta_{\text{state}}) - (y_{\text{state}} - y_{C_f})\cos(\theta_{\text{state}}) \\ \theta_1 &\leftarrow -\theta_{\text{state}} \end{aligned}$$

to every $(x_{\text{state}}, y_{\text{state}}, \theta_{\text{state}})$ slice-vertex in V_{state}, where (x_{C_f}, y_{C_f}) is the fixed-object center of uncertainty P_{C_f}. V_1 is the set of true locations of P_{C_f}, measured with respect to the moving-object origin, that insure that the (x, y, θ) configuration is in V_{state}.

Shrink V_1 by the fixed-object position uncertainty $(\epsilon_{xy_f}, \epsilon_{\theta_f})$ to form V_2, the set of nominal P_{C_f} locations that insure that the true P_{C_f} location is in V_1.

Form the volume V_3 by applying the transform

$$\begin{aligned} x_3 &\leftarrow x_f + x_{C_f} - x_2\cos(\theta_2) - y_2\sin(\theta_2) \\ y_3 &\leftarrow y_f + y_{C_f} + x_2\sin(\theta_2) - y_2\cos(\theta_2) \\ \theta_3 &\leftarrow \theta_f - \theta_2 \end{aligned}$$

to every (x_2, y_2, θ_2) slice-vertex in V_2. V_3 is the set of true locations of the moving-object origin, measured in the robot's coordinate frame, that insure that the nominal P_{C_f} location is in V_2.

Compensate for $\mathcal{U}_m$.
Form the volume V_4 by applying the transform

$$\begin{aligned} x_4 &\leftarrow x_3 + x_{C_m}\cos(\theta_3) - y_{C_m}\sin(\theta_3) \\ y_4 &\leftarrow y_3 + x_{C_m}\sin(\theta_3) + y_{C_m}\cos(\theta_3) \\ \theta_4 &\leftarrow \theta_3 \end{aligned}$$

to every (x_3, y_3, θ_3) slice-vertex in V_3, where (x_{C_m}, y_{C_m}) is the moving-object center of uncertainty P_{C_m}. V_4 is the set of true locations of P_{C_m} that insure that the true location of the moving-object origin is in V_3.

Shrink V_4 by the moving-object position uncertainty $(\epsilon_{xy_m}, \epsilon_{\theta_m})$ to form V_{command}, the set of nominal P_{C_m} locations that insure that the true P_{C_m} location is in V_4.

Return V_{command}.
If the robot commands the moving-object control point P_{C_m} to move to a position in V_{command}, the true (x, y, θ) configuration that results will be contained in V_{state}. In the above construction, V_{state} is measured relative to the fixed-object origin, V_1 and V_2 are measured relative to the moving-object origin, and V_3, V_4, and V_{command} are measured relative to the robot's coordinate frame.

Figure 177: The *COMMAND*$_{\text{moving}}$ procedure.

COMMAND$_{\text{fixed}}(V_{\text{state}}, \mathcal{U}_m, \mathcal{U}_f, x_m, y_m, \theta_m)$

V_{state} defines a set of true positions of the moving-object origin, measured relative to the true position of the fixed-object origin, and (x_m, y_m, θ_m) is the nominal position of the moving-object origin, measured relative to the robot's coordinate frame. We seek to transform V_{state} into a volume V_{command}, defining a set of points measured in the robot's coordinate frame, such that if the robot commands a motion of the fixed-object to a position in V_{command}, the resulting true position of the moving-object origin measured relative to the fixed-object origin is in V_{state}.

Compensate for $\mathcal{U}_m$.
Form the volume V_1 by applying the transform

$$\begin{aligned} x_1 &\leftarrow x_{\text{state}} + x_{C_m}\cos(\theta_{\text{state}}) - y_{C_m}\sin(\theta_{\text{state}}) \\ y_1 &\leftarrow y_{\text{state}} + x_{C_m}\sin(\theta_{\text{state}}) + y_{C_m}\cos(\theta_{\text{state}}) \\ \theta_1 &\leftarrow \theta_{\text{state}} \end{aligned}$$

to every $(x_{\text{state}}, y_{\text{state}}, \theta_{\text{state}})$ slice-vertex in V_{state}, where (x_{C_m}, y_{C_m}) is the moving-object center of uncertainty P_{C_m}. V_1 is the set of true locations of P_{C_m} that insure that the moving-object origin is in V_{state}.

Shrink V_1 by the moving-object position uncertainty $(\epsilon_{xy_m}, \epsilon_{\theta_m})$ to form V_2, the set of nominal P_{C_m} locations that insure that the true P_{C_m} location is in V_1.

Form the volume V_3 by applying the transform

$$\begin{aligned} x_3 &\leftarrow x_2 - x_{C_m}\cos(\theta_2) + y_{C_m}\sin(\theta_2) \\ y_3 &\leftarrow y_2 - x_{C_m}\sin(\theta_2) - y_{C_m}\cos(\theta_2) \\ \theta_3 &\leftarrow \theta_2 \end{aligned}$$

to every (x_2, y_2, θ_2) slice-vertex in V_2. V_3 is the set of nominal locations of the moving-object origin that insure that the nominal P_{C_m} location is in V_2.

Compensate for $\mathcal{U}_f$.
Form the volume V_4 by applying the transform

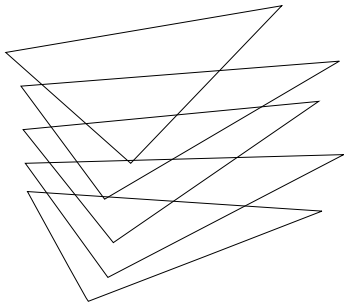
$$\begin{aligned} x_4 &\leftarrow x_m - (x_3 - x_{C_f})\cos(\theta_3) - (y_3 - y_{C_f})\sin(\theta_3) \\ y_4 &\leftarrow y_m + (x_3 - x_{C_f})\sin(\theta_3) - (y_3 - y_{C_f})\cos(\theta_3) \\ \theta_4 &\leftarrow \theta_m - \theta_3 \end{aligned}$$

to every (x_3, y_3, θ_3) slice-vertex in V_3, where (x_{C_f}, y_{C_f}) is the fixed-object center of uncertainty P_{C_f}. V_4 is the set of true locations of P_{C_f} that insure that the nominal location of the moving-object origin is in V_3.

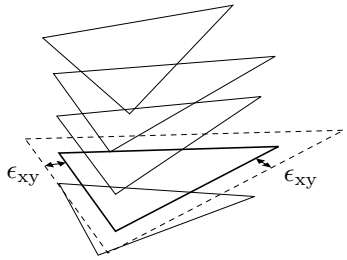
Shrink V_4 by the fixed-object position uncertainty $(\epsilon_{xy_f}, \epsilon_{\theta_f})$ to form V_{command}, the set of nominal P_{C_f} locations that insure that the true P_{C_f} location is in V_4.

Return V_{command}.
If the robot commands the fixed-object control point P_{C_f} to move to a position in V_{command}, the true (x, y, θ) configuration that results will be contained in V_{state}. In the above construction, V_{state}, V_1, V_2, and V_3 are measured relative to the fixed-object origin, and V_4 and V_{command} are measured relative to the robot's coordinate frame.

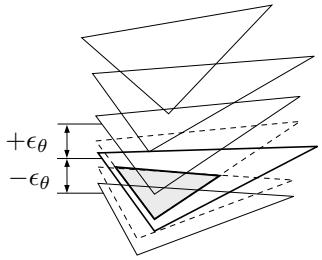
Figure 178: The *COMMAND*$_{\text{fixed}}$ procedure.



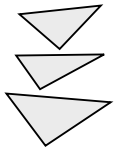
(a)



(b)



(c)



(d)

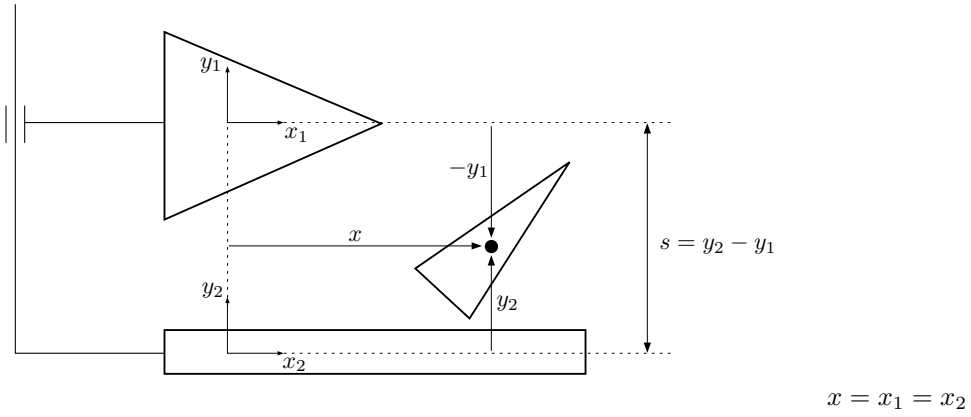
Figure 179: Shrinking a volume to account for uncertainty. (a) An initial volume. (b) After shrinking each slice by ϵ_{xy}. (c) Shrinking a given slice to account for the θ-uncertainty ϵ_θ; the slice is intersected with all adjacent slices within a distance ϵ_θ. (d) After shrinking all slices to account for ϵ_θ; this is the final shrunk volume.

Part III
Appendices

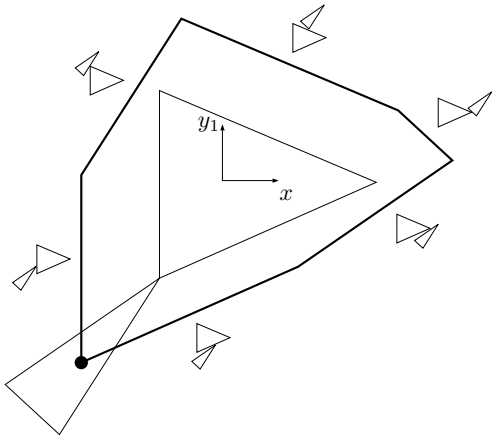
Appendix 1: Constructing Configuration Obstacles for Planar Grippers with Two Fingers

Figure 26 in Part I shows the configuration-obstacle for a two-fingered gripper. This obstacle may be constructed from the output of the *CO* algorithm, using a simple linear combination procedure.

Figure 180 shows this construction for the case where only translations are allowed. The construction relies on the observations that $s = y_2 - y_1$, and that when both fingers are in contact with the moving-polygon, y_1 and y_2 are not independent. This implies that the set of configurations corresponding to two-finger contacts may be constructed by a linear combination of the usual configuration obstacles for finger #1 and #2, as shown in Figure 180(e). The negated configuration obstacle for finger #1 is added to the configuration obstacle for finger #2. This addition is performed along the y-dimension, so the resulting curve is embedded in the $(x, y_2 - y_1)$ space, which is also the (x, s) space. Notice that this construction produces four curves; these correspond to all combinations of the upper and lower curves of the original obstacles. Of course, this linear combination is only defined over the x-values where both obstacles are defined.

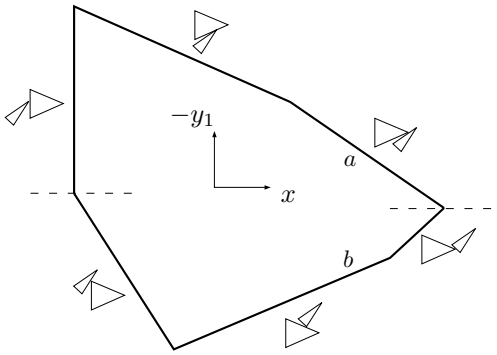


(a)

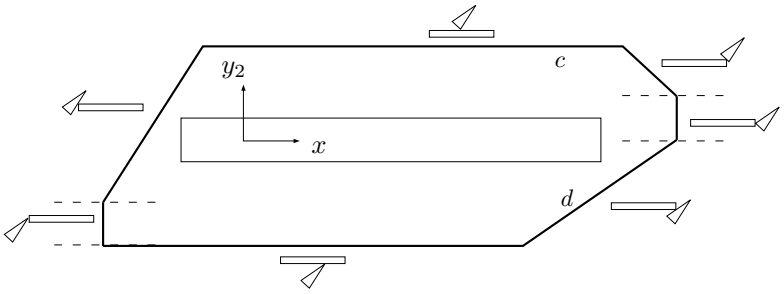


(b)

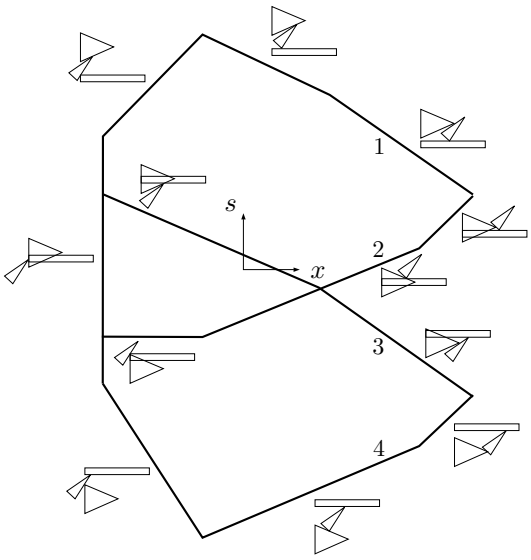
Figure 180: Constructing the set of two-finger contact configurations, without rotation. (a) Definition of coordinate systems. (b) The configuration-obstacle for finger #1, expressed in the (x, y_1) configuration space.



(c)



(d)



$1 = a + c$

$2 = b + c$

$3 = a + d$

$4 = b + d$

(e)

Figure 180 (continued): (c) The obstacle for finger #1, reflected about x. (d) The obstacle for finger #2, expressed in the (x, y_2) configuration space. (e) The two-finger configuration obstacle, expressed in the (x, s) configuration space. Each edge of this obstacle is a linear combination of an edge from (c) and (d).

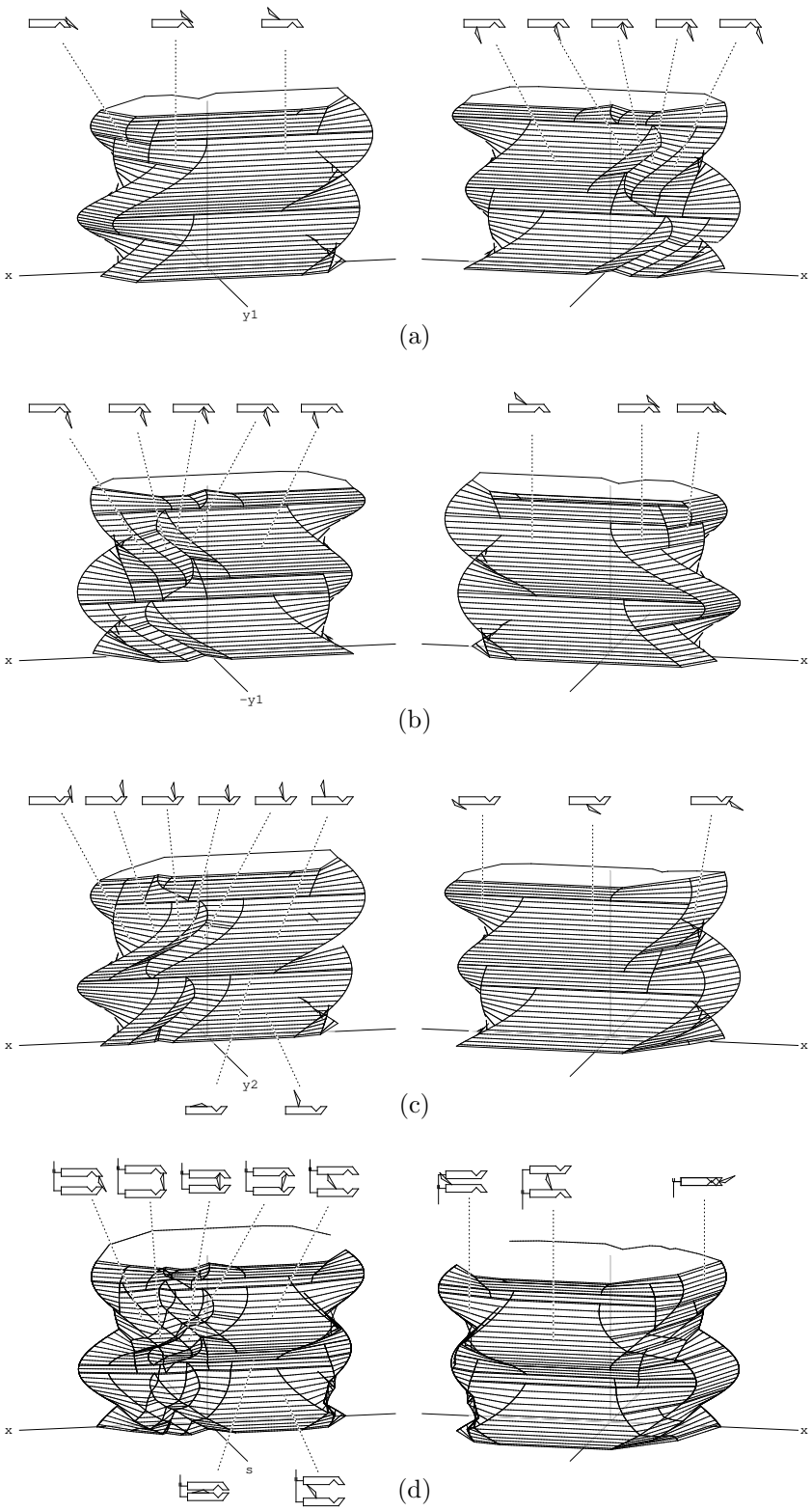


Figure 181: Constructing the set of two-finger contact configurations, including rotation. (a) The finger #1 obstacle, expressed in the (x, y_1, θ) configuration space. (b) The finger #1 obstacle, negated about y_1. (c) The finger #2 obstacle, expressed in the (x, y_2, θ) configuration space. (d) The two-finger configuration obstacle, expressed in the (x, s, θ) configuration space. Only two of the four surfaces are shown.

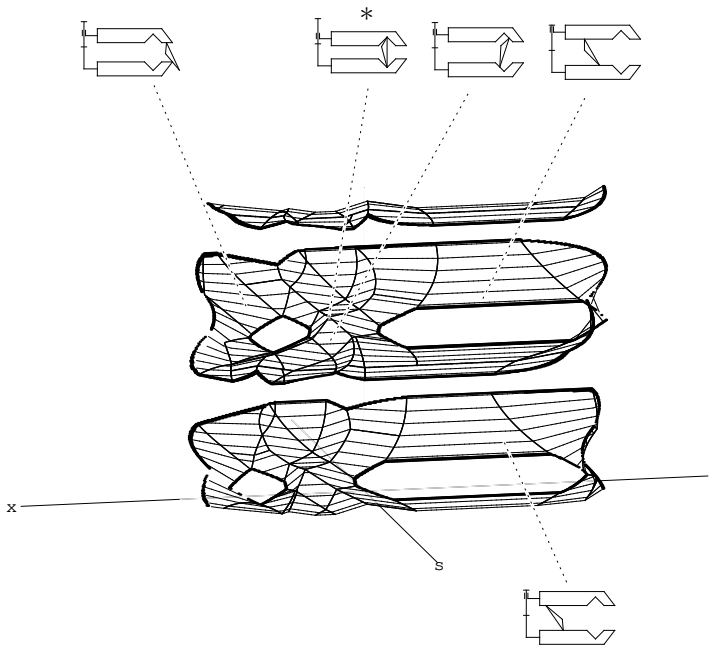


Figure 182: The effect of gripper travel limits. Reachable two-finger configurations must lie between the $s = s_{\min}$ and $s = s_{\max}$ planes in the (x, s, θ) space.

This construction can also be applied when rotations are allowed, as shown in Figure 181. Here the linear combination procedure is applied between obstacle surfaces, producing a surface embedded in an (x, s, θ) configuration space. The facets of this surface correspond to intersections of facets from the original configuration obstacles, after projection along the collinear y_1 and y_2 axes. Figure 182 shows the effect of adding travel limits to the gripper; only configurations within an interval of s values are reachable.

It may also be possible to apply this linear combination procedure to revolute and pantograph grippers by first transforming the finger configuration-space obstacles into the appropriate coordinate spaces (Figure 183). For example, to form the set of two-finger contact configurations for a revolute gripper, we could construct the (x, y_i, θ) configuration-obstacles for each finger, transform them into the appropriate (x, σ_i, θ) spaces, reflect obstacle #1 about its σ_1 axis, and form the linear combination as before.

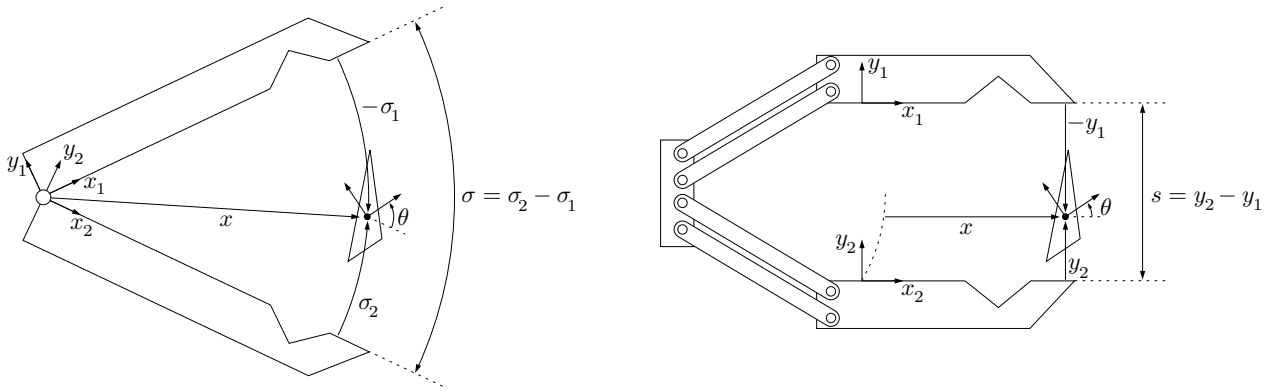


Figure 183: Coordinate systems for revolute and pantograph gripper configurations.

This suggests the following procedure for constructing the set of two-finger contact configurations for planar grippers comprised of a pair of fingers connected by a one-degree-of-freedom linkage:

1. Construct the configuration-space obstacle of each finger using the *CO* algorithm.

2. Transform each obstacle into the appropriate (x, var, θ) parameter space, where *var* is a variable aligned with the motion of the linkage connecting the fingers.

3. Reflect obstacle #1 about its *var* axis.

4. Split the obstacle surfaces into subsurfaces of points whose constant-θ normals have either positive, negative, or zero dot-products with the local *var* axis direction.

5. Assemble all pairs of subsurfaces (S_1, S_2) that have either positive or negative dot-products, and apply the following:

    a. Project S_1 and S_2 along the projection direction onto the (x, θ) plane, producing S_1' and S_2'.

    b. Intersect the facets of S_1' and S_2', forming a collection of composite facets.

    c. For each composite facet F', set the *var* metric fields of F' to reflect the sum of the *var* metric fields of the S_1 and S_2 facets that intersected to form F'.

6. Using the connectivity of the original obstacles and the metric coincidences between the transformed surfaces, connect the zero-dot-product subsurfaces to the subsurfaces constructed in the previous step to form a topologically correct two-finger obstacle surface.

7. If gripper travel limits are present, cut the two-finger surface at the $var_{\min}$ and $var_{\max}$ planes.

This high-level procedure description neglects many important details, which are areas for future research. The figures shown in this thesis were not generated by this algorithm; instead, they were generated by a similar algorithm that rendered the two-contact configuration obstacle surface using a very slow point-by-point linear combination process.

The significance of this construction is that it supports subsequent grasp analysis. Neglecting situations where one-finger grasp configurations are of interest, the surface shown in Figure 182 corresponds to all possible grasp configurations between a planar robot gripper and a manipulated object. By applying a friction analysis similar to the *STATIC* algorithm, one could identify all possible stable grasp configurations. Grasp planning could be then performed by algorithms similar to the BP_e and BP_i algorithms.

For example, the label marked * in Figure 182 indicates a stable grasp lying in a concavity of the obstacle surface; a modified version of the BP_e algorithm could identify a "puddle" of surrounding configurations that would be guaranteed to achieve this grasp configuration, as long as the gripper control servo is designed to assure that s decreases monotonically. Extending the BP_i algorithm to this domain is more challenging, since representations for free, single-contact, and two-contact configurations must be used in concert to produce proper physical predictions.

Appendix 2: Finding the Roots of Trigonometric Polynomials

This appendix presents a general technique for solving equations of the form:

$$0 = k_0 + k_1 \mathrm{s}_\theta + k_2 \mathrm{c}_\theta + k_3 \mathrm{c}_\theta \mathrm{s}_\theta + k_4 \mathrm{s}_\theta^2 + k_5 \mathrm{c}_\theta^2 + k_6 \mathrm{c}_\theta \mathrm{s}_\theta^2 + k_7 \mathrm{c}_\theta^2 \mathrm{s}_\theta + k_8 \mathrm{s}_\theta^3 + k_9 \mathrm{c}_\theta^3 + \cdots$$

where s_θ and c_θ are abbreviations for $\sin(\theta)$ and $\cos(\theta)$, respectively. This technique may be used to identify the θ-roots of arbitrary polynomials of s_θ and c_θ. We solve these polynomials by the following steps:

1. Eliminate s_θ from the polynomial, producing a new polynomial of just c_θ. This elimination doubles the degree of the original polynomial.
2. Find the roots of the new polynomial, using a recursive-derivative/binary-search numerical procedure. The resulting c_θ values are roots of the new polynomial.
3. Convert the c_θ values to θ values, using the inverse cosine function. In general, this produces two θ values for every c_θ root; half of these θ values are spurious.
4. Substitute the θ values into the original polynomial to identify and discard spurious values.
5. Return the remaining θ roots.

This procedure always finds all of the θ-roots of the input polynomial, and never returns θ values that are not roots. These steps are further explained below.

Eliminating s_θ

We remove s_θ terms from the polynomial by applying the identity $\mathrm{s}_\theta^2 = 1 - \mathrm{c}_\theta^2$ until all s_θ terms have degree 1 or 0. This gives:

$$0 = k'_0 + k'_1 \mathrm{s}_\theta + k'_2 \mathrm{c}_\theta + k'_3 \mathrm{c}_\theta \mathrm{s}_\theta + k'_4 \mathrm{c}_\theta^2 + k'_5 \mathrm{c}_\theta^2 \mathrm{s}_\theta + k'_6 \mathrm{c}_\theta^3 + \cdots$$

Collecting s_θ terms gives:

$$\mathrm{s}_\theta = \frac{k'_0 + k'_2 \mathrm{c}_\theta + k'_4 \mathrm{c}_\theta^2 + k'_6 \mathrm{c}_\theta^3 + \cdots}{-k'_1 - k'_3 \mathrm{c}_\theta - k'_5 \mathrm{c}_\theta^2 - \cdots}$$

We then square both sides of this equation, and apply the identity $\mathrm{s}_\theta^2 = 1 - \mathrm{c}_\theta^2$ again. This produces an equation of just c_θ, which can be re-arranged to a polynomial of the form:

$$0 = k''_0 + k''_1 \mathrm{c}_\theta + k''_2 \mathrm{c}_\theta^2 + k''_3 \mathrm{c}_\theta^3 + k''_4 \mathrm{c}_\theta^4 + k''_5 \mathrm{c}_\theta^5 + k''_6 \mathrm{c}_\theta^6 + \cdots$$

The next section describes a numerical procedure for finding the roots of equations in this class. In the algorithms reported in this thesis, all of this algebra was performed off-line; the algorithms compute the polynomial coefficients $k''_0, k''_1, \ldots$ directly, given the desired edge-intersection or critical-force specification.

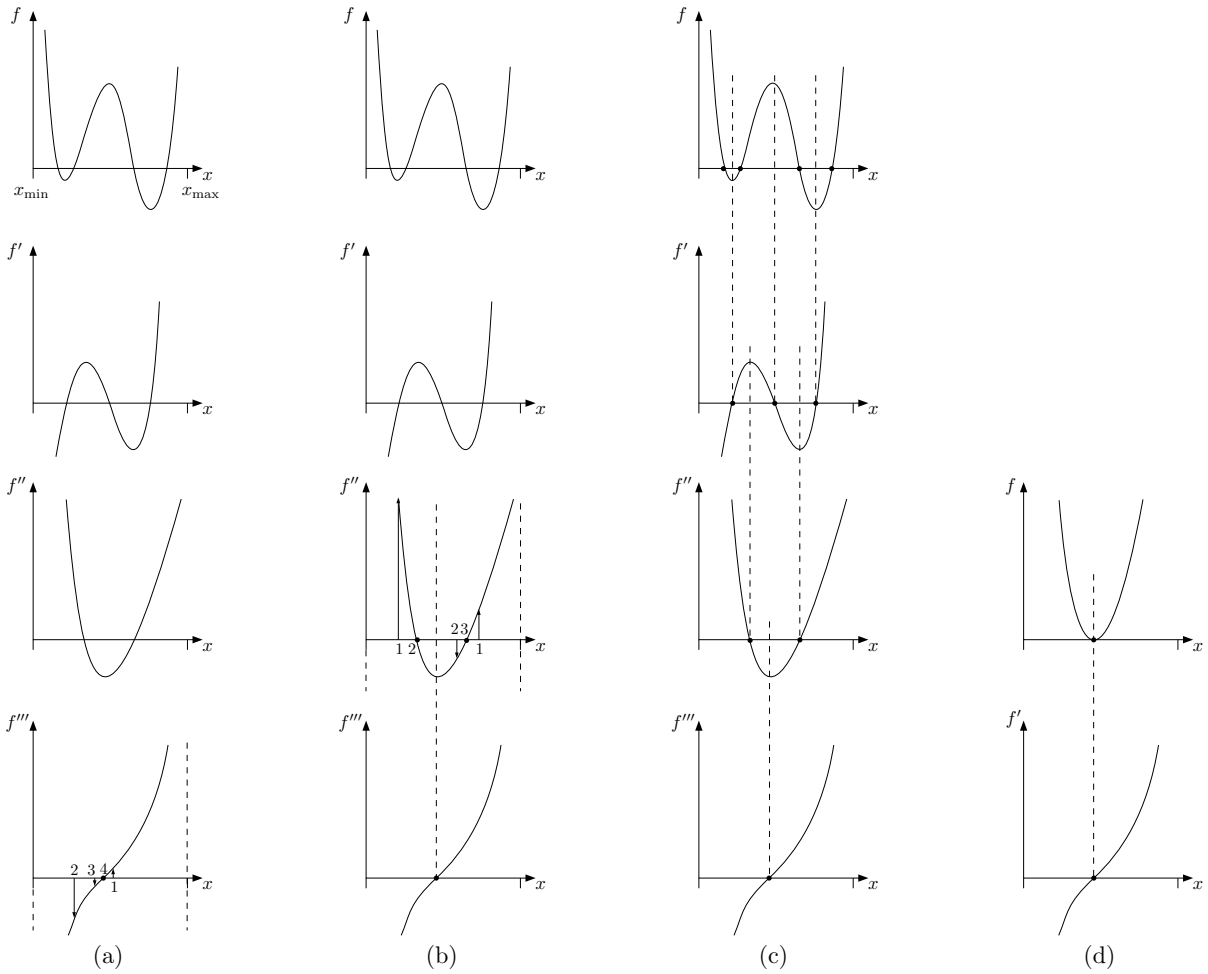


Figure 184: Finding the roots of a fourth-degree polynomial $y = f(x)$, where all roots of interest lie in the interval $[x_{\min}, x_{\max}]$. The upper plot shows the polynomial $f(x)$, and the plots below it show the derivatives $f'(x)$, $f''(x)$, and $f'''(x)$. Each successive derivative has one less root than its predecessor. (a) The third derivative $f'''(x)$ has at most one root; this may be found using binary search with $x_{\min}$ and $x_{\max}$ as starting points. (b) This root is then used to split $[x_{\min}, x_{\max}]$ into two parts; the second derivative $f''(x)$ may have at most one root in each part. Binary search is again applied to find the roots of $f''(x)$. (c) This process is repeated until the roots of the original polynomial are found. (d) One of several special cases that can arise.

Finding c_θ Roots Numerically

The transformed c_θ polynomial has the following two useful characteristics: First, all of its roots must be in the interval $[-1, 1]$, since it is a polynomial of $\cos(\theta)$. Second, it has n or fewer roots, where n is the highest exponent appearing in the polynomial.

We can numerically find all of the roots of a polynomial within a given interval by recursively finding the roots of the polynomial derivatives, and using these roots and the bounding interval to constrain subsequent binary search operations. Figure 184 shows an example of this procedure applied to a fourth-degree polynomial.

This procedure may be used to find the roots of polynomials of arbitrary degree, as long as we are only interested in roots that appear within some bounded interval. In our case, all c_θ roots must appear between -1 and 1; one could imagine other scenarios where arbitrary roots are possible, but only the roots within a given window are of interest.

Discarding Spurious Roots

The generated θ values reflect the ambiguity of the inverse cosine function: every c_θ root gives rise to two θ values. Half of these θ values are spurious; we can remove these by substituting each θ value into the original polynomial, evaluating the result, and retaining only the θ values that produce zero results.

This process is complicated by the presence of numerical errors. Because several numerical transformations have been applied, the θ values generated thus far may differ from the true θ roots. Consequently, a local search procedure is applied to numerically improve the accuracy of each root. If no true θ' root can be found near a given θ value, then the θ value is discarded; otherwise, the improved θ' value is returned.

Tolerance Scaling

If given an interval where (i) the function values at each endpoint have opposite sign, and (ii) there is at most one root in the interval, numerical binary search will reliably find the root of any continuous function. Further, this root may be refined to an arbitrary precision, without the use of fuzzy arithmetic.

However, some roots are not contained in such an interval; the special case appearing in Figure 184(d) is an example. Our procedure will identify this root, but it must be recognized by observing that the endpoint of a binary search interval produces a zero function value. To maintain consistency with the fuzzy arithmetic employed by other geometric procedures, this condition must be recognized using fuzzy arithmetic.

This requires scaling of the tolerance ϵ to reflect the transformation applied to the original polynomial to produce the c_θ polynomial. Since the coefficients of the c_θ polynomial are typically much larger than the coefficients of the original polynomial, using the original tolerance to detect roots of the c_θ polynomial essentially amounts to applying a more restrictive tolerance, which could lead to later inconsistencies.

To prevent this problem, we will scale the tolerance ϵ to reflect the transformation. The appropriate scaling function varies with the form of the polynomial, and is determined by carrying ϵ through the polynomial transformation. A simple example is shown below; similar derivations are used for other forms. We begin with the fuzzy version of the root equation:

$$\epsilon > |k_0 + k_1 s_\theta + k_2 c_\theta|$$

The worst-case bounds of this equation are:

$$\pm\epsilon = k_0 + k_1 s_\theta + k_2 c_\theta$$

We eliminate s_θ by applying the polynomial transformation, giving:

$$\begin{aligned} -k_1 s_\theta &= (k_0 \pm \epsilon) + k_2 c_\theta \\ s_\theta^2 &= \left[\frac{(k_0 \pm \epsilon) + k_2 c_\theta}{-k_1}\right]^2 \\ \pm \underbrace{\epsilon[2(k_0 + k_2 c_\theta) + \epsilon]}_{\epsilon''} &= \underbrace{(k_0^2 - k_1^2)}_{k_0''} + \underbrace{2k_0 k_2}_{k_1''} c_\theta + \underbrace{(k_1^2 + k_2^2)}_{k_2''} c_\theta^2 \end{aligned}$$

Since c_θ is bounded by 1 and -1, ϵ is scaled by the maximum of $[2(k_0 + k_2) + \epsilon]$ and $[2(k_0 - k_2) + \epsilon]$.

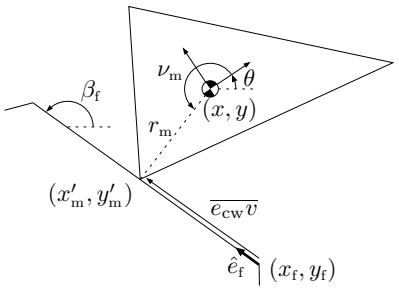
Appendix 3: Configuration Obstacle Feature Equations

This appendix presents the equations describing configuration-obstacle features. Derivations are omitted for brevity; see Figure 63 for an example derivation. Throughout this appendix, s_α and c_α are abbreviations for $\sin(\alpha)$ and $\cos(\alpha)$.

3.1. Facets

There are two types of obstacle facets, arising from ve and ev contacts. For each type, we require the functions $f_x(p,\theta)$, $f_y(p,\theta)$, and $f_p(x,y,\theta)$ to describe the unbounded facet c-surface.

VE Facets

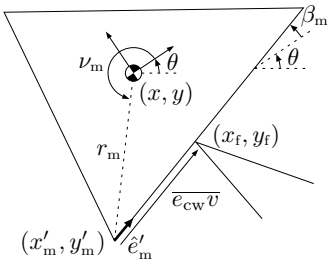


$$f_x(p,\theta) \longleftarrow x_\mathrm{f} + p\mathrm{c}_{\beta_\mathrm{f}} - r_\mathrm{m}\mathrm{c}_{(\theta+\nu_\mathrm{m})}$$

$$f_y(p,\theta) \longleftarrow y_\mathrm{f} + p\mathrm{s}_{\beta_\mathrm{f}} - r_\mathrm{m}\mathrm{s}_{(\theta+\nu_\mathrm{m})}$$

$$f_p(x,y,\theta) \longleftarrow \left\{ \begin{array}{lcl} x'_\mathrm{m} & \leftarrow & x + r_\mathrm{m}\mathrm{c}_{(\theta+\nu_\mathrm{m})} \\ y'_\mathrm{m} & \leftarrow & y + r_\mathrm{m}\mathrm{s}_{(\theta+\nu_\mathrm{m})} \\ \overline{e_\mathrm{cw}v} & \leftarrow & \left[\begin{array}{c} x'_\mathrm{m} - x_\mathrm{f} \\ y'_\mathrm{m} - y_\mathrm{f} \end{array} \right] \\ \text{return} & & \overline{e_\mathrm{cw}v} \cdot \hat{e}_\mathrm{f} \end{array} \right.$$

EV Facets



$$f_x(p,\theta) \longleftarrow x_\mathrm{f} - p\mathrm{c}_{(\theta+\beta_\mathrm{m})} - r_\mathrm{m}\mathrm{c}_{(\theta+\nu_\mathrm{m})}$$

$$f_y(p,\theta) \longleftarrow y_\mathrm{f} - p\mathrm{s}_{(\theta+\beta_\mathrm{m})} - r_\mathrm{m}\mathrm{s}_{(\theta+\nu_\mathrm{m})}$$

$$f_p(x,y,\theta) \longleftarrow \left\{ \begin{array}{lcl} x'_\mathrm{m} & \leftarrow & x + r_\mathrm{m}\mathrm{c}_{(\theta+\nu_\mathrm{m})} \\ y'_\mathrm{m} & \leftarrow & y + r_\mathrm{m}\mathrm{s}_{(\theta+\nu_\mathrm{m})} \\ \overline{e_\mathrm{cw}v} & \leftarrow & \left[\begin{array}{c} x_\mathrm{f} - x'_\mathrm{m} \\ y_\mathrm{f} - y'_\mathrm{m} \end{array} \right] \\ \hat{e}'_\mathrm{m} & \leftarrow & \left[\begin{array}{c} \mathrm{c}_{(\theta+\beta_\mathrm{m})} \\ \mathrm{s}_{(\theta+\beta_\mathrm{m})} \end{array} \right] \\ \text{return} & & \overline{e_\mathrm{cw}v} \cdot \hat{e}'_\mathrm{m} \end{array} \right.$$

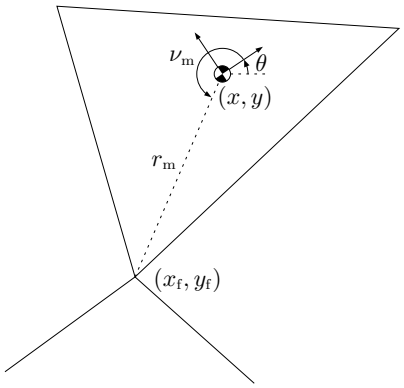
3.2. Edges

There are six types of obstacle edges: ee, vv, ve-ve, ev-ev, ve-ev, and multiple-class-0. Since multiple-class-0 edges are compositions of other edge types, special equations are not required for constructing these edges. For the remaining edge types, we require functions that describe the metric properties of the unbounded curve containing the edge. These curves fall into two broad categories: class-0 and non-class-0 curves. Most edge types can give rise to both types of curves, depending on the geometry of the contact features (see Figure 62).

Class-0 curves are straight lines that lie in a constant-θ plane; we describe these curves with a θ-value and three linear functions: $f_x(p)$, $f_y(p)$, and $f_p(x, y)$. The basic construction of these functions is identical for the various edge types: In each case, the contact geometry is used to identify the θ-value of the edge, and then this θ-value and the contact geometry are used to generate the appropriate linear functions. This appendix omits these functions because they are straightforward but verbose.

Non-class-0 curves are functions of θ, and fall into three classes. The following tables present the equations of these curves; the more complex cases include a few derivation steps in addition to the final equation.

VV Edges



$$x = x_{\mathrm{f}} - r_{\mathrm{m}}\mathrm{c}_{(\theta+\nu_{\mathrm{m}})}$$

$$f_x(\theta) \leftarrow k_{0_x} + k_{1_x}\mathrm{s}_\theta + k_{2_x}\mathrm{c}_\theta$$

where

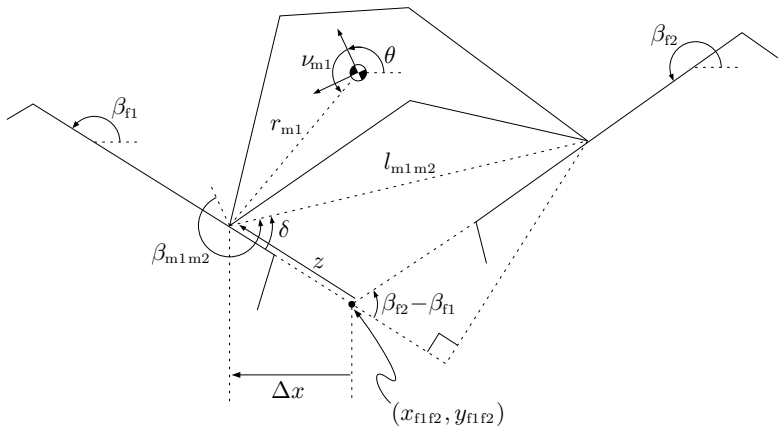
$$\begin{aligned} k_{0_x} &= x_{\mathrm{f}} \\ k_{1_x} &= y_{\mathrm{m}} \\ k_{2_x} &= -x_{\mathrm{m}} \end{aligned}$$

$$f_y(\theta) \leftarrow k_{0_y} + k_{1_y}\mathrm{s}_\theta + k_{2_y}\mathrm{c}_\theta$$

where

$$\begin{aligned} k_{0_y} &= y_{\mathrm{f}} \\ k_{1_y} &= -x_{\mathrm{m}} \\ k_{2_y} &= -y_{\mathrm{m}} \end{aligned}$$

VE-VE Edges



$$
\begin{aligned}
x &= x_{\mathrm{f1f2}} + \Delta x - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + \mathrm{c}_{\beta_{\mathrm{f1}}} z - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + \mathrm{c}_{\beta_{\mathrm{f1}}} \left[l_{\mathrm{m1m2}}\mathrm{c}_{\delta} - \frac{l_{\mathrm{m1m2}}\mathrm{s}_{\delta}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{\mathrm{f1f2}} + l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}} \left[\mathrm{c}_{(\theta+\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\theta+\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})}
\end{aligned}
$$

$$
f_x(\theta) \leftarrow k_{0_x} + k_{1_x}\mathrm{s}_{\theta} + k_{2_x}\mathrm{c}_{\theta}
$$

where

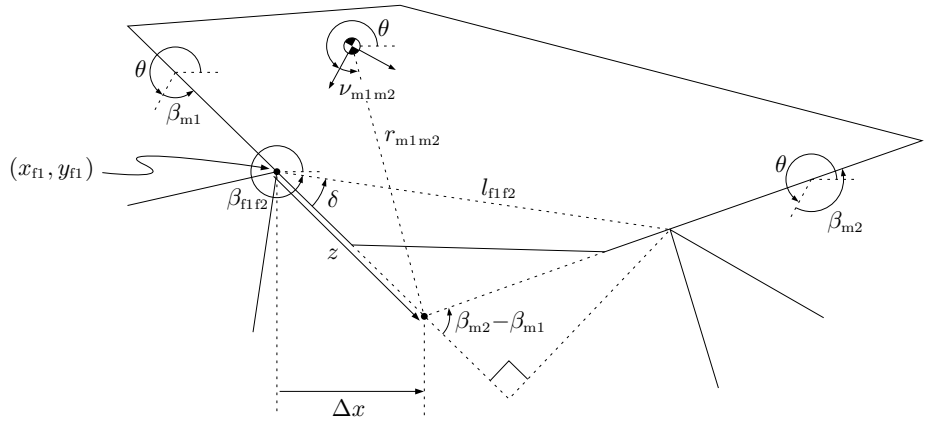
$$
\begin{aligned}
k_{0_x} &= x_{\mathrm{f1f2}} \\
k_{1_x} &= l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}} \left(-\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right) + y_{\mathrm{m1}} \\
k_{2_x} &= l_{\mathrm{m1m2}}\mathrm{c}_{\beta_{\mathrm{f1}}} \left(\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right) - x_{\mathrm{m1}}
\end{aligned}
$$

$$
f_y(\theta) \leftarrow k_{0_y} + k_{1_y}\mathrm{s}_{\theta} + k_{2_y}\mathrm{c}_{\theta}
$$

where

$$
\begin{aligned}
k_{0_y} &= y_{\mathrm{f1f2}} \\
k_{1_y} &= l_{\mathrm{m1m2}}\mathrm{s}_{\beta_{\mathrm{f1}}} \left(-\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right) - x_{\mathrm{m1}} \\
k_{2_y} &= l_{\mathrm{m1m2}}\mathrm{s}_{\beta_{\mathrm{f1}}} \left(\mathrm{c}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)} - \frac{\mathrm{s}_{(\beta_{\mathrm{m1m2}}-\beta_{\mathrm{f1}}-\pi)}}{\tan(\beta_{\mathrm{f2}} - \beta_{\mathrm{f1}})} \right) - y_{\mathrm{m1}}
\end{aligned}
$$

EV-EV Edges



$$
\begin{aligned}
x &= x_{\mathrm{f1}} + \Delta x - r_{\mathrm{m1m2}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1m2}})} \\
x &= x_{\mathrm{f1}} + \mathrm{c}_{(\theta+\beta_{\mathrm{m1}})} z - r_{\mathrm{m1m2}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1m2}})} \\
x &= x_{\mathrm{f1}} + \mathrm{c}_{(\theta+\beta_{\mathrm{m1}})} \left[l_{\mathrm{f1f2}}\mathrm{c}_{\delta} - \frac{l_{\mathrm{f1f2}}\mathrm{s}_{\delta}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] - r_{\mathrm{m1m2}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1m2}})} \\
x &= x_{\mathrm{f1}} + \mathrm{c}_{(\theta+\beta_{\mathrm{m1}})} \left[l_{\mathrm{f1f2}}\mathrm{c}_{(\beta_{\mathrm{f1f2}}-\theta-\beta_{\mathrm{m1}})} - \frac{l_{\mathrm{f1f2}}\mathrm{s}_{(\beta_{\mathrm{f1f2}}-\theta-\beta_{\mathrm{m1}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] - r_{\mathrm{m1m2}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1m2}})}
\end{aligned}
$$

$$f_x(\theta) \leftarrow k_{0_x} + k_{1_x}\mathrm{s}_{\theta} + k_{2_x}\mathrm{c}_{\theta} + k_{3_x}\mathrm{c}_{\theta}\mathrm{s}_{\theta} + k_{4_x}\mathrm{s}_{\theta}^2$$

where

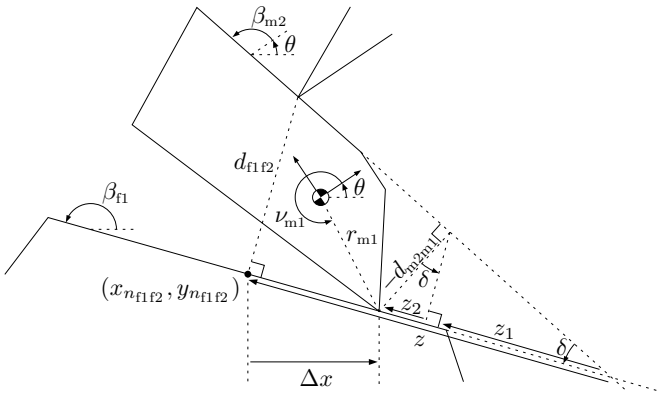
$$
\begin{aligned}
k_{0_x} &= x_{\mathrm{f1}} + l_{\mathrm{f1f2}}\mathrm{c}_{\beta_{\mathrm{m1}}} \left[\mathrm{c}_{(\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{s}_{(\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] \\
k_{1_x} &= y_{\mathrm{m1m2}} \\
k_{2_x} &= -x_{\mathrm{m1m2}} \\
k_{3_x} &= l_{\mathrm{f1f2}} \left[-\mathrm{s}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{c}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] \\
k_{4_x} &= -l_{\mathrm{f1f2}} \left[\mathrm{c}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{s}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right]
\end{aligned}
$$

$$f_y(\theta) \leftarrow k_{0_y} + k_{1_y}\mathrm{s}_{\theta} + k_{2_y}\mathrm{c}_{\theta} + k_{3_y}\mathrm{c}_{\theta}\mathrm{s}_{\theta} + k_{4_y}\mathrm{s}_{\theta}^2$$

where

$$
\begin{aligned}
k_{0_y} &= y_{\mathrm{f1}} + l_{\mathrm{f1f2}}\mathrm{s}_{\beta_{\mathrm{m1}}} \left[\mathrm{c}_{(\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{s}_{(\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] \\
k_{1_y} &= -x_{\mathrm{m1m2}} \\
k_{2_y} &= -y_{\mathrm{m1m2}} \\
k_{3_y} &= l_{\mathrm{f1f2}} \left[\mathrm{c}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{s}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right] \\
k_{4_y} &= l_{\mathrm{f1f2}} \left[-\mathrm{s}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})} + \frac{\mathrm{c}_{(2\beta_{\mathrm{m1}}-\beta_{\mathrm{f1f2}})}}{\tan(\beta_{\mathrm{m2}} - \beta_{\mathrm{m1}})} \right]
\end{aligned}
$$

VE-EV Edges



$$
\begin{aligned}
x &= x_{n_{\mathrm{f1f2}}} + \Delta x - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{n_{\mathrm{f1f2}}} - \mathrm{c}_{\beta_{\mathrm{f1}}}[z - z_1 - z_2] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{n_{\mathrm{f1f2}}} - \mathrm{c}_{\beta_{\mathrm{f1}}}\left[z + z\frac{d_{\mathrm{m2m1}}\mathrm{c}_{\delta}}{d_{\mathrm{f1f2}}} + d_{\mathrm{m2m1}}\mathrm{s}_{\delta}\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{n_{\mathrm{f1f2}}} - \mathrm{c}_{\beta_{\mathrm{f1}}}\left[\frac{d_{\mathrm{f1f2}}}{\tan(\delta)}\left(1 + \frac{d_{\mathrm{m2m1}}\mathrm{c}_{\delta}}{d_{\mathrm{f1f2}}}\right) + d_{\mathrm{m2m1}}\mathrm{s}_{\delta}\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})} \\
x &= x_{n_{\mathrm{f1f2}}} - \mathrm{c}_{\beta_{\mathrm{f1}}}\left[\frac{d_{\mathrm{f1f2}}}{\tan(\beta_{\mathrm{f1}} - \beta_{\mathrm{m2}} - \theta)}\left(1 + \frac{d_{\mathrm{m2m1}}\mathrm{c}_{(\beta_{\mathrm{f1}}-\beta_{\mathrm{m2}}-\theta)}}{d_{\mathrm{f1f2}}}\right) + d_{\mathrm{m2m1}}\mathrm{s}_{(\beta_{\mathrm{f1}}-\beta_{\mathrm{m2}}-\theta)}\right] - r_{\mathrm{m1}}\mathrm{c}_{(\theta+\nu_{\mathrm{m1}})}
\end{aligned}
$$

$$
f_x(\theta) \leftarrow k_{0_x} + k_{1_x}\mathrm{s}_{\theta} + k_{2_x}\mathrm{c}_{\theta} + \frac{k_{3_x} + k_{4_x}\mathrm{s}_{\theta} + k_{5_x}\mathrm{c}_{\theta}}{k_{6_x}\mathrm{s}_{\theta} + k_{7_x}\mathrm{c}_{\theta}}
$$

where

$$
\begin{aligned}
k_{0_x} &= x_{n_{\mathrm{f1f2}}} \\
k_{1_x} &= y_{\mathrm{m1}} \\
k_{2_x} &= -x_{\mathrm{m1}} \\
k_{3_x} &= d_{\mathrm{m2m1}}\mathrm{c}_{\beta_{\mathrm{f1}}} \\
k_{4_x} &= -d_{\mathrm{f1f2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\mathrm{s}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{5_x} &= d_{\mathrm{f1f2}}\mathrm{c}_{\beta_{\mathrm{f1}}}\mathrm{c}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{6_x} &= \mathrm{c}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{7_x} &= \mathrm{s}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})}
\end{aligned}
$$

$$
f_y(\theta) \leftarrow k_{0_y} + k_{1_y}\mathrm{s}_{\theta} + k_{2_y}\mathrm{c}_{\theta} + \frac{k_{3_y} + k_{4_y}\mathrm{s}_{\theta} + k_{5_y}\mathrm{c}_{\theta}}{k_{6_y}\mathrm{s}_{\theta} + k_{7_y}\mathrm{c}_{\theta}}
$$

where

$$
\begin{aligned}
k_{0_y} &= y_{n_{\mathrm{f1f2}}} \\
k_{1_y} &= -x_{\mathrm{m1}} \\
k_{2_y} &= -y_{\mathrm{m1}} \\
k_{3_y} &= d_{\mathrm{m2m1}}\mathrm{s}_{\beta_{\mathrm{f1}}} \\
k_{4_y} &= -d_{\mathrm{f1f2}}\mathrm{s}_{\beta_{\mathrm{f1}}}\mathrm{s}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{5_y} &= d_{\mathrm{f1f2}}\mathrm{s}_{\beta_{\mathrm{f1}}}\mathrm{c}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{6_y} &= \mathrm{c}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})} \\
k_{7_y} &= \mathrm{s}_{(\beta_{\mathrm{m2}}-\beta_{\mathrm{f1}})}
\end{aligned}
$$

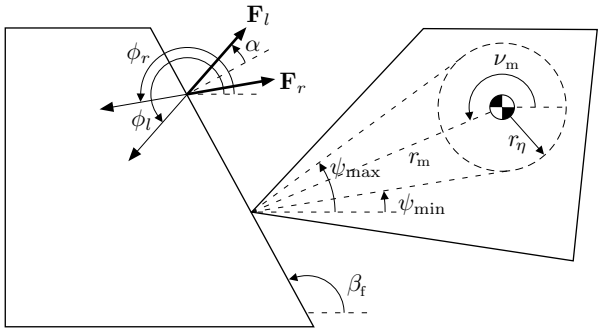
Appendix 4: Static Analysis Equations

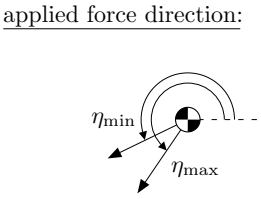
4.1. Facets

To identify the subset of a configuration obstacle facet where equilibrium is possible, the *STATIC* algorithm identifies the subset of the facet c-surface where equilibrium is possible, and then intersects this set with the obstacle facet to identify the reachable possible-equilibrium configurations. This section describes the methods used to construct the possible-equilibrium subset of a facet c-surface for both ve and ev facets. In this section, $\alpha = \arctan(\mu_{\max})$ is the half-angle of the largest possible friction cone for the facet contact.

VE Facets

The subset of a ve c-surface where static equilibrium is possible may be empty, the entire c-surface, or a pair of horizontal bands that span the full length of the c-surface. The construction of this subset is simple, and is illustrated by the pseudo-code procedure given below.





construct-facet-possible-equilibrium-subset$_{\mathrm{ve}}(F, \mathcal{F}_a)$

$\phi_r \leftarrow \beta_f + \frac{\pi}{2} - \alpha$

$\phi_l \leftarrow \beta_f + \frac{\pi}{2} + \alpha$

$[\sigma_{\min}, \sigma_{\max}] \leftarrow [\phi_r, \phi_l] \cap [\eta_{\min}, \eta_{\max}]$

If $[\sigma_{\min}, \sigma_{\max}] = \emptyset$, then

    Static equilibrium is never possible.

else

    If $r_{\mathrm{m}} \leq r_\eta$, then

        The contact vertex lies within the applied force point uncertainty ball;
        static equilibrium is always possible.

    else

        $\psi_o \leftarrow \nu_{\mathrm{m}} - \pi$

        $\delta\psi \leftarrow \arcsin\left(\frac{r_\eta}{r_{\mathrm{m}}}\right)$

        $\psi_{\min} \leftarrow \psi_o - \delta\psi$

        $\psi_{\max} \leftarrow \psi_o + \delta\psi$

        Static equilibrium is possible for $\theta \in [\sigma_{\min} - \psi_{\max}, \sigma_{\max} - \psi_{\min}]$ and
        $\theta \in [\sigma_{\min} - \psi_{\max} + \pi, \sigma_{\max} - \psi_{\min} + \pi]$.

    endif

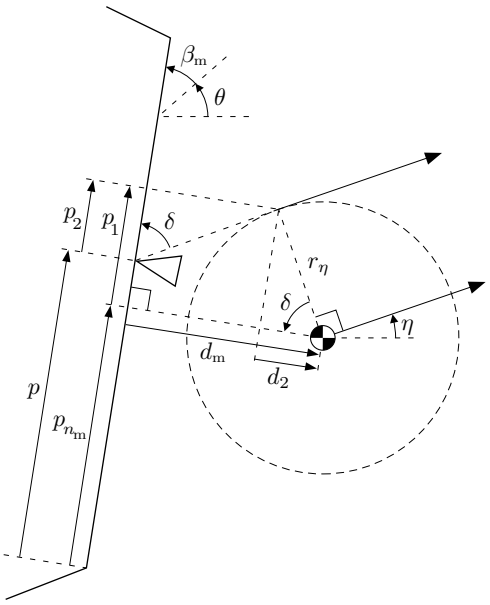
endif

EV Facets

The subset of an ev c-surface where static equilibrium is possible is always a single closed, bounded, non-null region. The shape of this region varies from a single point to an area bounded by up to 10 boundary segments. The region shape depends on the friction coefficient α, the contact geometry, and the applied force uncertainty parameters r_η and $\epsilon_\eta = \eta_{\max} - \eta_{\min}$. When $\alpha = r_\eta = \epsilon_\eta = 0$, the possible-equilibrium subset is an isolated point; in all other cases the possible-equilibrium subset is a finite continuous set.

Because the shape of the possible-equilibrium region depends on several interacting parameters, the procedure that constructs the region is a fairly opaque and tedious case analysis. Instead of presenting this procedure here, the following pages illustrate a series of example possible-equilibrium regions, demonstrating the contributions of each parameter. The first page illustrates three possible-equilibrium regions for cases with varying contact geometry, without uncertainty. The pages that follow illustrate the contributions of the uncertainty parameters r_η and ϵ_η. For clarity, these examples show the possible-equilibrium region plotted in the (p, θ) space that parameterizes the facet c-surface, instead of showing the region drawn on the curved helicoidal c-surface in the (x, y, θ) space.

All of the boundary segments that border the possible-equilibrium region of an ev c-surface are either line segments that correspond to a constant p or θ value, or non-linear functions of θ that correspond to a coincidence between the boundary of the applied force uncertainty ball and an applied-force ray projected through the contact point. The equation that describes the non-linear boundary segments is derived below.



$$\tan(\delta) = \frac{d_m - d_2}{p_2}$$

$$p_2 = \frac{d_m - d_2}{\tan(\delta)}$$

$$\delta = \theta + \beta_m - \eta$$

$$p = p_{n_m} + p_1 - p_2$$

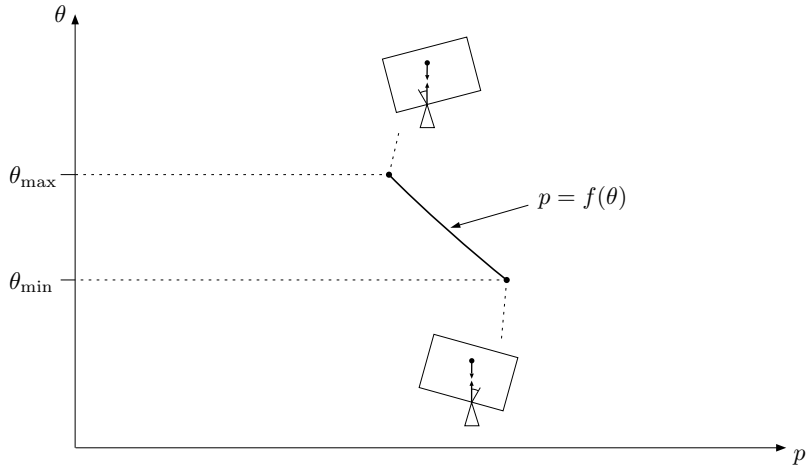
$$p = p_{n_m} + r_\eta \mathrm{s}_\delta - \frac{d_m - d_2}{\tan(\delta)}$$

$$p = p_{n_m} + r_\eta \mathrm{s}_\delta - \frac{d_m - r_\eta \mathrm{c}_\delta}{\tan(\delta)}$$

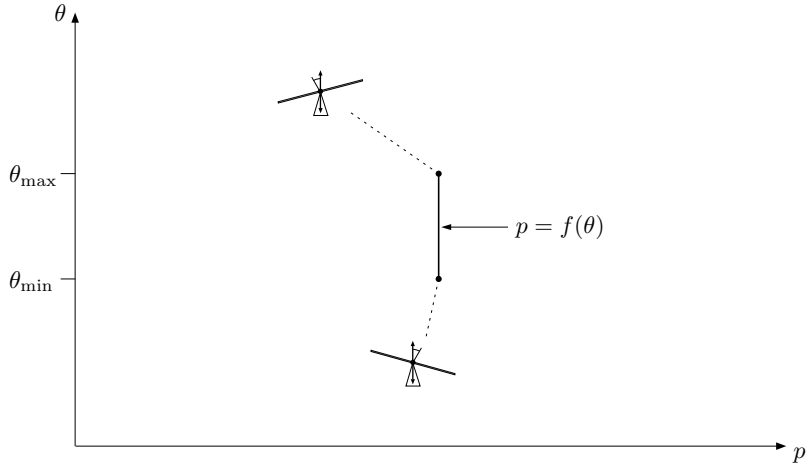
$$\boxed{p = p_{n_m} + r_\eta \mathrm{s}_{(\theta+\beta_m-\eta)} - \frac{d_m - r_\eta \mathrm{c}_{(\theta+\beta_m-\eta)}}{\tan(\theta+\beta_m-\eta)}}$$

$\alpha > 0$, $r_\eta = 0$, $\epsilon_\eta = 0$

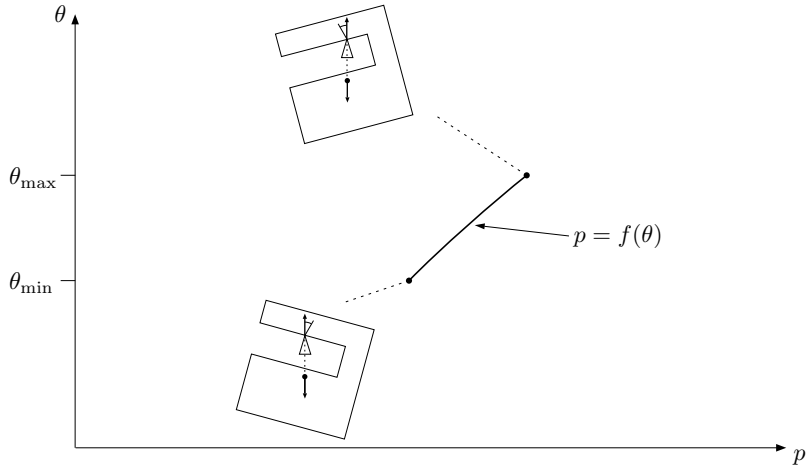
$d < 0$



$d = 0$



$d > 0$

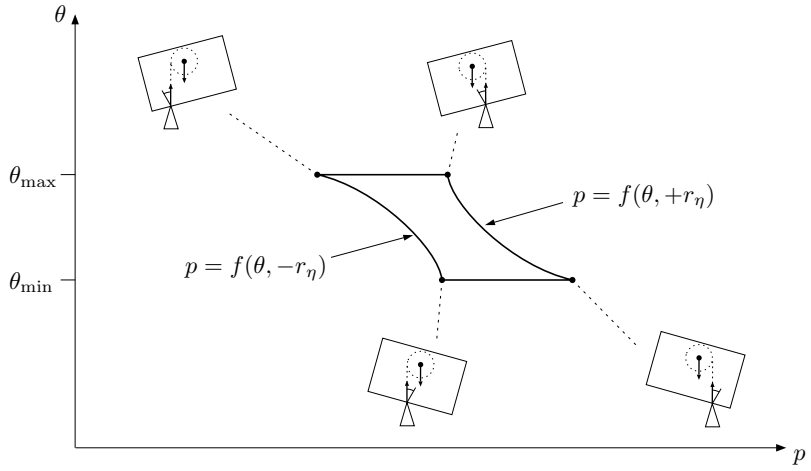


$$f(\theta) = p_{n_\mathrm{m}} - \frac{d_\mathrm{m}}{\tan(\theta+\beta_\mathrm{m}-\eta)}$$

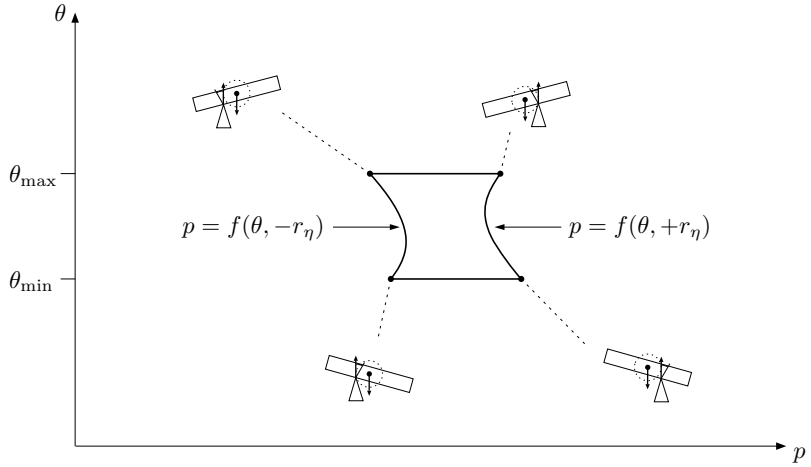
$$\theta_\mathrm{min} = \eta + \tfrac{\pi}{2} - \beta_\mathrm{m} - \alpha$$
$$\theta_\mathrm{max} = \eta + \tfrac{\pi}{2} - \beta_\mathrm{m} + \alpha$$

$\alpha > 0$, $r_\eta > 0$, $\epsilon_\eta = 0$

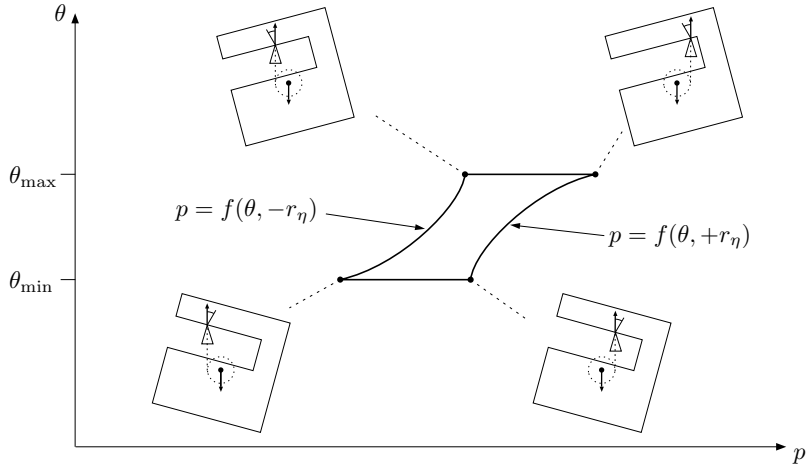
$d < -r_\eta$



$|d| < r_\eta$



$d > r_\eta$

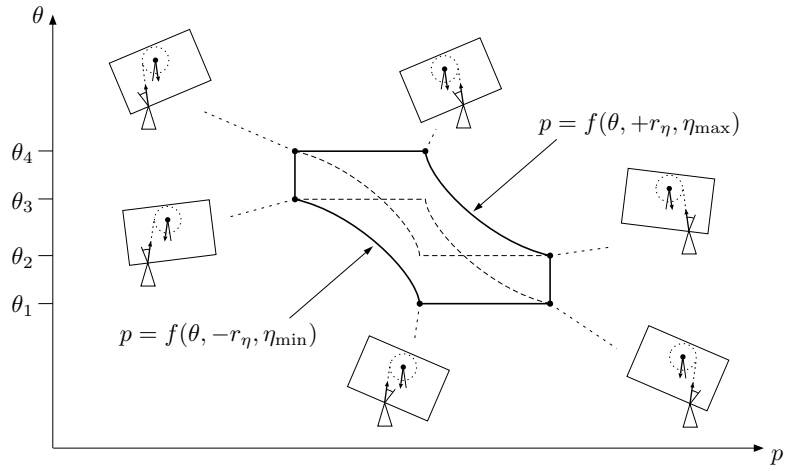


$$f(\theta, r) = p_{n_\mathrm{m}} + r\mathrm{s}_{(\theta+\beta_\mathrm{m}-\eta)} - \frac{d_\mathrm{m} - r\mathrm{c}_{(\theta+\beta_\mathrm{m}-\eta)}}{\tan(\theta+\beta_\mathrm{m}-\eta)}$$

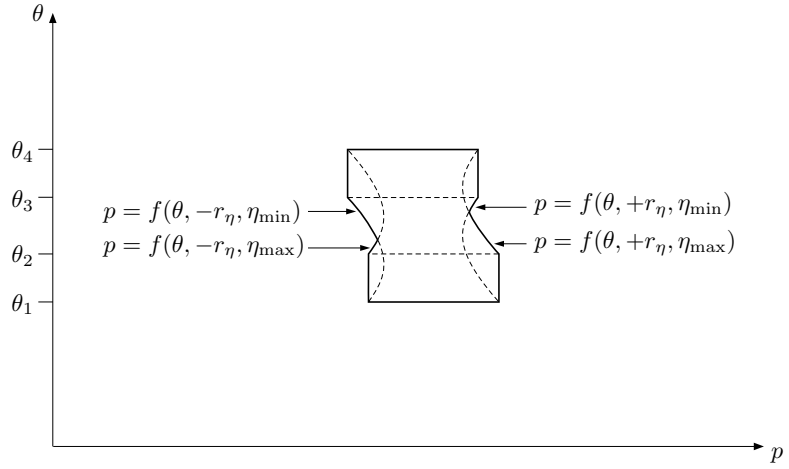
$$\begin{aligned} \theta_{\min} &= \eta + \tfrac{\pi}{2} - \beta_\mathrm{m} - \alpha \\ \theta_{\max} &= \eta + \tfrac{\pi}{2} - \beta_\mathrm{m} + \alpha \end{aligned}$$

$\alpha > 0$, $r_\eta > 0$, $\epsilon_\eta > 0$

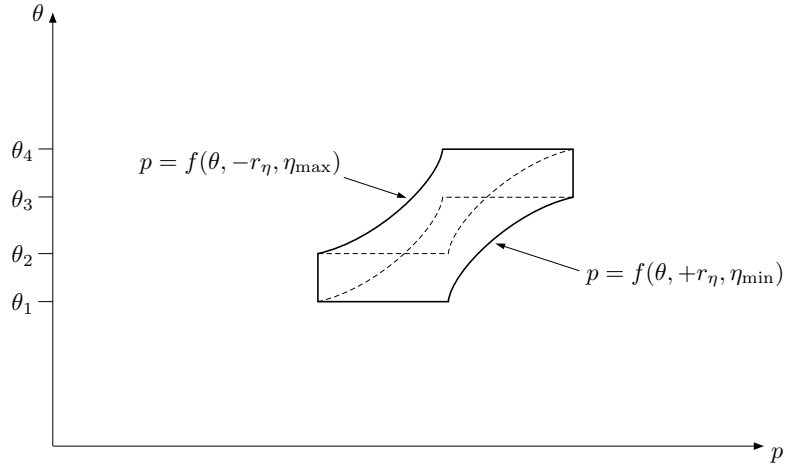
$d < -r_\eta$



$|d| < r_\eta$



$d > r_\eta$



$$f(\theta, r, \eta) = p_{n_\mathrm{m}} + r\mathrm{s}_{(\theta+\beta_\mathrm{m}-\eta)} - \frac{d_\mathrm{m} - r\mathrm{c}_{(\theta+\beta_\mathrm{m}-\eta)}}{\tan(\theta+\beta_\mathrm{m}-\eta)}$$

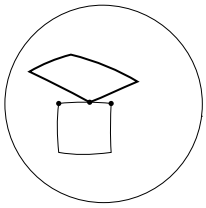
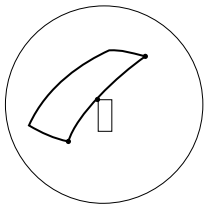
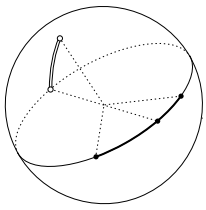
$$\begin{aligned}
\theta_1 &= \eta_{\min} + \tfrac{\pi}{2} - \beta_\mathrm{m} - \alpha \\
\theta_2 &= \eta_{\max} + \tfrac{\pi}{2} - \beta_\mathrm{m} - \alpha \\
\theta_3 &= \eta_{\min} + \tfrac{\pi}{2} - \beta_\mathrm{m} + \alpha \\
\theta_4 &= \eta_{\max} + \tfrac{\pi}{2} - \beta_\mathrm{m} + \alpha
\end{aligned}$$

4.2. Edges

To identify the subset of a configuration obstacle edge where equilibrium is possible, the *STATIC* algorithm identifies critical values of the edge parameter p or θ that correspond to topologically critical conditions on the force sphere. These critical values are then analyzed to construct intervals along the obstacle edge where static equilibrium is either always possible, or never possible. This section presents the equations used to identify these critical values.

These equations identify all edge parameter values where a critical condition occurs between a force-sphere vertex and the great circle containing a force-sphere edge; subsequent processing then considers the edge endpoints to decide whether the condition is actually critical. When implementing the *STATIC* algorithm, I tested each equation using an interactive computer program that displayed each constructed great circle critical condition; this program allowed me to verify the correctness of the equations on a variety of test cases.

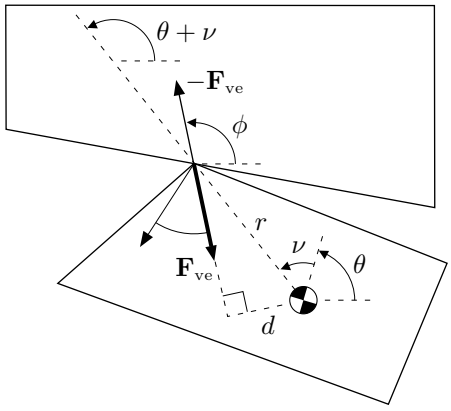
Because there are several types of obstacle edges and also several types of critical conditions, a large number of equations are required to construct each possible critical condition. These cases are illustrated in the table below, and the corresponding equations are given on the following pages. Class-0 cases are omitted for brevity; their derivation is straightforward but verbose due to the bookkeeping required to maintain a consistent coordinate system. Note that for multiple-class-0 edges, the class-0 analysis must consider all combinations of friction cone rays to avoid missing critical p values.

Critical condition	VV	VE-VE	EV-EV	VE-EV
	Page 276	Page 276 (VV Case 1)	Page 280	Page 282 (table)
	Page 277	Page 279	Page 281	Page 282 (table)
	Impossible	Page 279	Dual of VE-VE	Page 282 (table)

Preliminaries

We begin by defining the terms used in the static analysis equations, and develop the basic expressions for the force-sphere points corresponding to ve and ev friction cone rays. In the derivations that follow, $\alpha = \arctan(\mu_{\max})$, which is the half-angle of the largest possible friction cone for a given contact. The angles ϕ and γ describe the directions of negated ve and ev friction-cone rays, respectively. ϕ is measured relative to the fixed-object's coordinate frame, while γ is measured relative to the moving object's coordinate frame. The expressions for ve and ev friction cone rays are shown below; these are derived by substituting the appropriate terms into equation (1), given on page 153.

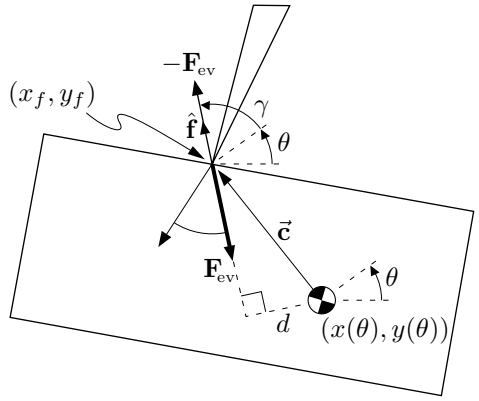
VE-ray



$$-\mathbf{F}_{\text{ve}}(\theta) \quad = \quad \begin{bmatrix} \text{c}_{\phi} \\ \text{s}_{\phi} \\ -\frac{r}{\rho}\text{s}_{(\theta+\nu-\phi)} \end{bmatrix}$$

$d = r\text{s}_{(\theta+\nu-\phi)}$

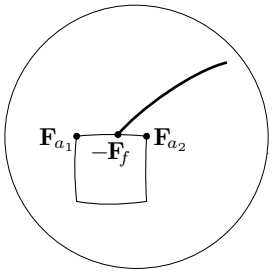
EV-ray



$$-\mathbf{F}_{\text{ev}}(\theta) \quad = \quad \begin{bmatrix} \text{c}_{(\theta+\gamma)} \\ \text{s}_{(\theta+\gamma)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\text{s}_{(\theta+\gamma)} - (y_f - y(\theta))\,\text{c}_{(\theta+\gamma)}\right] \end{bmatrix}$$

$d = |\hat{\mathbf{f}} \times \vec{\mathbf{c}}| = (y_f - y(\theta))\,\text{c}_{(\theta+\gamma)} - (x_f - x(\theta))\,\text{s}_{(\theta+\gamma)}$

VV Case 1: Fixed normal direction



$$0 = -\mathbf{F}_f(\theta) \cdot [\mathbf{F}_{a_1} \times \mathbf{F}_{a_2}]$$

$$0 = \begin{bmatrix} \mathrm{c}_\phi \\ \mathrm{s}_\phi \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-\phi)} \end{bmatrix} \cdot \underbrace{\left(\begin{bmatrix} x_{a_1} \\ y_{a_1} \\ z_{a_1} \end{bmatrix} \times \begin{bmatrix} x_{a_2} \\ y_{a_2} \\ z_{a_2} \end{bmatrix} \right)}_{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

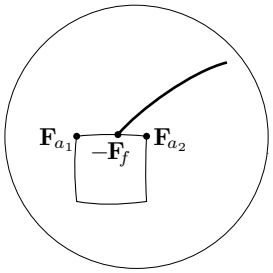
where

$$k_0 = x_A\mathrm{c}_\phi + y_A\mathrm{s}_\phi$$

$$k_1 = -\frac{r}{\rho}z_A\mathrm{c}_{(\nu-\phi)}$$

$$k_2 = -\frac{r}{\rho}z_A\mathrm{s}_{(\nu-\phi)}$$

VV Case 2: Varying normal direction



$$0 = -\mathbf{F}_f(\theta) \cdot [\mathbf{F}_{a_1} \times \mathbf{F}_{a_2}]$$

$$0 = \begin{bmatrix} \mathrm{c}_{(\theta+\gamma)} \\ \mathrm{s}_{(\theta+\gamma)} \\ -\frac{r}{\rho}\mathrm{s}_{(\nu-\gamma)} \end{bmatrix} \cdot \underbrace{\left(\begin{bmatrix} x_{a_1} \\ y_{a_1} \\ z_{a_1} \end{bmatrix} \times \begin{bmatrix} x_{a_2} \\ y_{a_2} \\ z_{a_2} \end{bmatrix} \right)}_{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

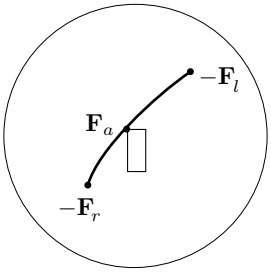
where

$$k_0 = -\frac{r}{\rho}z_A\mathrm{s}_{(\nu-\gamma)}$$

$$k_1 = -x_A\mathrm{s}_\gamma + y_A\mathrm{c}_\gamma$$

$$k_2 = x_A\mathrm{c}_\gamma + y_A\mathrm{s}_\gamma$$

VV Case 3: Both rays have fixed normal direction



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_r(\theta) \times -\mathbf{F}_l(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{\phi_r} \\ \mathrm{s}_{\phi_r} \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-\phi_r)} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\phi_r+2\alpha)} \\ \mathrm{s}_{(\phi_r+2\alpha)} \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-(\phi_r+2\alpha))} \end{bmatrix} \right)$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

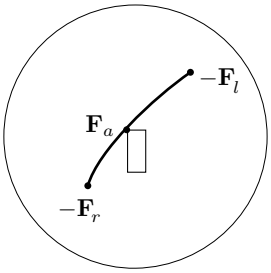
where

$$k_0 = z_a$$

$$k_1 = \frac{r}{\rho}(x_a\mathrm{c}_\nu + y_a\mathrm{s}_\nu)$$

$$k_2 = \frac{r}{\rho}(x_a\mathrm{s}_\nu - y_a\mathrm{c}_\nu)$$

VV Case 4: One ray has fixed normal direction, and the other has varying normal direction



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_r(\theta) \times -\mathbf{F}_l(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{\phi} \\ \mathrm{s}_{\phi} \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-\phi)} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\theta+\gamma)} \\ \mathrm{s}_{(\theta+\gamma)} \\ -\frac{r}{\rho}\mathrm{s}_{(\nu-\gamma)} \end{bmatrix} \right)$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta + k_3\mathrm{c}_\theta\mathrm{s}_\theta + k_4\mathrm{s}_\theta^2 + k_5\mathrm{c}_\theta^2$$

where

$$k_0 = \frac{r}{\rho}(-x_a\mathrm{s}_\phi\mathrm{s}_{(\nu-\gamma)} + y_a\mathrm{c}_\phi\mathrm{s}_{(\nu-\gamma)})$$

$$k_1 = z_a(\mathrm{c}_\gamma\mathrm{c}_\phi + \mathrm{s}_\gamma\mathrm{s}_\phi)$$

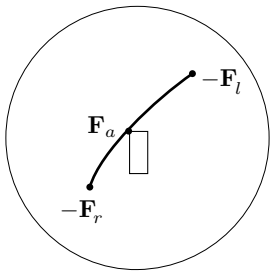
$$k_2 = z_a(\mathrm{s}_\gamma\mathrm{c}_\phi - \mathrm{c}_\gamma\mathrm{s}_\phi)$$

$$k_3 = \frac{r}{\rho}\left[x_a(\mathrm{c}_\gamma\mathrm{s}_{(\nu-\phi)} + \mathrm{s}_\gamma\mathrm{c}_{(\nu-\phi)}) + y_a(\mathrm{s}_\gamma\mathrm{s}_{(\nu-\phi)} - \mathrm{c}_\gamma\mathrm{c}_{(\nu-\phi)}\right]$$

$$k_4 = \frac{r}{\rho}(x_a\mathrm{c}_\gamma\mathrm{c}_{(\nu-\phi)} + y_a\mathrm{s}_\gamma\mathrm{c}_{(\nu-\phi)})$$

$$k_5 = \frac{r}{\rho}(x_a\mathrm{s}_\gamma\mathrm{s}_{(\nu-\phi)} - y_a\mathrm{c}_\gamma\mathrm{s}_{(\nu-\phi)})$$

VV Case 5: Both rays have varying normal direction



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_r(\theta) \times -\mathbf{F}_l(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{(\theta+\gamma_r)} \\ \mathrm{s}_{(\theta+\gamma_r)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma_r)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma_r)}\right] \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\theta+\gamma_r+2\alpha)} \\ \mathrm{s}_{(\theta+\gamma_r+2\alpha)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma_r+2\alpha)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma_r+2\alpha)}\right] \end{bmatrix} \right)$$

where $\gamma_r + 2\alpha = \gamma_l$. Simplifying the cross-product gives:

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\mathrm{s}_{2\alpha} \begin{bmatrix} \frac{1}{\rho}(y_f - y(\theta)) \\ -\frac{1}{\rho}(x_f - x(\theta)) \\ 1 \end{bmatrix} \right)$$

Substituting $x(\theta) = k_{0_x} + k_{1_x}\mathrm{s}_\theta + k_{2_x}\mathrm{c}_\theta$ and $y(\theta) = k_{0_y} + k_{1_y}\mathrm{s}_\theta + k_{2_y}\mathrm{c}_\theta$ gives:

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

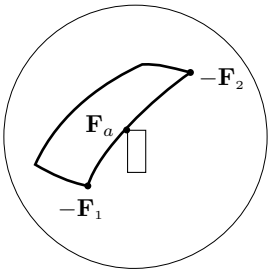
where

$$k_0 = \frac{1}{\rho}\left[y_a(k_{0_x} - x_f) - x_a(k_{0_y} - y_f)\right] + z_a$$

$$k_1 = \frac{1}{\rho}(y_a k_{1_x} - x_a k_{1_y})$$

$$k_2 = \frac{1}{\rho}(y_a k_{2_x} - x_a k_{2_y})$$

VE-VE



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_1(\theta) \times -\mathbf{F}_2(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{\phi_1} \\ \mathrm{s}_{\phi_1} \\ -\frac{r_1}{\rho}\mathrm{s}_{(\theta+\nu_1-\phi_1)} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{\phi_2} \\ \mathrm{s}_{\phi_2} \\ -\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-\phi_2)} \end{bmatrix} \right)$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

where

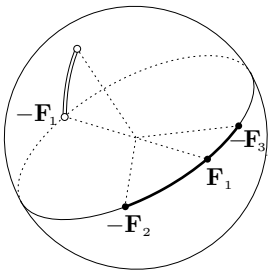
$$k_0 = z_a \left(\mathrm{c}_{\phi_1}\mathrm{s}_{\phi_2} - \mathrm{c}_{\phi_2}\mathrm{s}_{\phi_1}\right)$$

$$k_1 = x_a \tfrac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)} - x_a \tfrac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} + y_a \tfrac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{c}_{(\nu_2-\phi_2)} - y_a \tfrac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{c}_{(\nu_1-\phi_1)}$$

$$k_2 = x_a \tfrac{r_1}{\rho}\mathrm{s}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)} - x_a \tfrac{r_2}{\rho}\mathrm{s}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} + y_a \tfrac{r_2}{\rho}\mathrm{c}_{\phi_1}\mathrm{s}_{(\nu_2-\phi_2)} - y_a \tfrac{r_1}{\rho}\mathrm{c}_{\phi_2}\mathrm{s}_{(\nu_1-\phi_1)}$$

If $-\mathbf{F}_1$ and $-\mathbf{F}_2$ correspond to a single contact point, then this equation may be simplified to the equation given on the top of page 277, since $r_1 = r_2$, $\nu_1 = \nu_2$, and $\phi_2 = \phi_1 + 2\alpha$.

VE-VE



$$0 = \mathbf{F}_1(\theta) \cdot [-\mathbf{F}_2(\theta) \times -\mathbf{F}_3(\theta)]$$

We only need to consider cases where $-\mathbf{F}_2$ and $-\mathbf{F}_3$ are from the same contact, and $\mathbf{F}_1$ is from a different contact (other combinations are equivalent):

$$0 = \mathbf{F}_1(\theta) \cdot [-\mathbf{F}_{2r}(\theta) \times -\mathbf{F}_{2l}(\theta)]$$

$$0 = \begin{bmatrix} -\mathrm{c}_{\phi_1} \\ -\mathrm{s}_{\phi_1} \\ \frac{r_1}{\rho}\mathrm{s}_{(\theta+\nu_1-\phi_1)} \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{\phi_{2r}} \\ \mathrm{s}_{\phi_{2r}} \\ -\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-\phi_{2r})} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\phi_{2r}+2\alpha)} \\ \mathrm{s}_{(\phi_{2r}+2\alpha)} \\ -\frac{r_2}{\rho}\mathrm{s}_{(\theta+\nu_2-(\phi_{2r}+2\alpha))} \end{bmatrix} \right)$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta$$

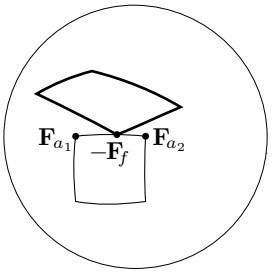
where

$$k_0 = 0$$

$$k_1 = -r_1\mathrm{c}_{(\nu_1-\phi_1)} - r_2\mathrm{c}_{(\nu_2-\phi_1)}$$

$$k_2 = -r_1\mathrm{s}_{(\nu_1-\phi_1)} - r_2\mathrm{s}_{(\nu_2-\phi_1)}$$

EV-EV



$$0 = -\mathbf{F}_f(\theta) \cdot [\mathbf{F}_{a_1} \times \mathbf{F}_{a_2}]$$

$$0 = \begin{bmatrix} \mathrm{c}_{(\theta+\gamma)} \\ \mathrm{s}_{(\theta+\gamma)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma)}\right] \end{bmatrix} \cdot \underbrace{\left(\begin{bmatrix} x_{a_1} \\ y_{a_1} \\ z_{a_1} \end{bmatrix} \times \begin{bmatrix} x_{a_2} \\ y_{a_2} \\ z_{a_2} \end{bmatrix} \right)}_{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}$$

where

$$\begin{aligned} x(\theta) &= k_{0_x} + k_{1_x}\mathrm{s}_\theta + k_{2_x}\mathrm{c}_\theta + k_{3_x}\mathrm{c}_\theta\mathrm{s}_\theta + k_{4_x}\mathrm{s}_\theta^2 \\ y(\theta) &= k_{0_y} + k_{1_y}\mathrm{s}_\theta + k_{2_y}\mathrm{c}_\theta + k_{3_y}\mathrm{c}_\theta\mathrm{s}_\theta + k_{4_y}\mathrm{s}_\theta^2 \end{aligned}$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta + k_3\mathrm{c}_\theta\mathrm{s}_\theta + k_4\mathrm{s}_\theta^2 + k_5\mathrm{c}_\theta^2 + k_6\mathrm{c}_\theta\mathrm{s}_\theta^2 + k_7\mathrm{c}_\theta^2\mathrm{s}_\theta + k_8\mathrm{s}_\theta^3$$

where

$$k_0 = 0$$

$$k_1 = -\tfrac{z_A}{\rho}\left[(k_{0_x} - x_f)\mathrm{c}_\gamma + (k_{0_y} - y_f)\mathrm{s}_\gamma\right] + y_A\mathrm{c}_\gamma - x_A\mathrm{s}_\gamma$$

$$k_2 = -\tfrac{z_A}{\rho}\left[(k_{0_x} - x_f)\mathrm{s}_\gamma + (k_{0_y} - y_f)\mathrm{c}_\gamma\right] + y_A\mathrm{s}_\gamma + x_A\mathrm{c}_\gamma$$

$$k_3 = -\tfrac{z_A}{\rho}\left[k_{1_x}\mathrm{s}_\gamma - k_{1_y}\mathrm{c}_\gamma + k_{2_x}\mathrm{c}_\gamma + k_{2_y}\mathrm{s}_\gamma\right]$$

$$k_4 = -\tfrac{z_A}{\rho}\left[k_{1_x}\mathrm{c}_\gamma + k_{1_y}\mathrm{s}_\gamma\right]$$

$$k_5 = -\tfrac{z_A}{\rho}\left[k_{2_x}\mathrm{s}_\gamma - k_{2_y}\mathrm{c}_\gamma\right]$$

$$k_6 = -\tfrac{z_A}{\rho}\left[k_{4_x}\mathrm{s}_\gamma - k_{4_y}\mathrm{c}_\gamma + k_{3_x}\mathrm{c}_\gamma + k_{3_y}\mathrm{s}_\gamma\right]$$

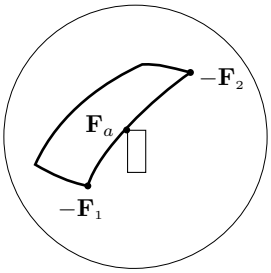
$$k_7 = -\tfrac{z_A}{\rho}\left[k_{3_x}\mathrm{s}_\gamma - k_{3_y}\mathrm{c}_\gamma\right]$$

$$k_8 = -\tfrac{z_A}{\rho}\left[k_{4_x}\mathrm{c}_\gamma + k_{4_y}\mathrm{s}_\gamma\right]$$

Since $k_{2_x} = -x_{\mathrm{m1m2}}$, $k_{1_x} = y_{\mathrm{m1m2}}$, $k_{2_y} = -y_{\mathrm{m1m2}}$, and $k_{1_y} = -x_{\mathrm{m1m2}}$, the term k_3 is always zero. This equation may be simplified for vertical edges of the applied force polygon, since $z_A = 0$.

EV-EV

$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_1(\theta) \times -\mathbf{F}_2(\theta)]$$



$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{(\theta+\gamma_1)} \\ \mathrm{s}_{(\theta+\gamma_1)} \\ \frac{1}{\rho}\left[(x_{f_1} - x(\theta))\,\mathrm{s}_{(\theta+\gamma_1)} - (y_{f_1} - y(\theta))\,\mathrm{c}_{(\theta+\gamma_1)}\right] \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\theta+\gamma_2)} \\ \mathrm{s}_{(\theta+\gamma_2)} \\ \frac{1}{\rho}\left[(x_{f_2} - x(\theta))\,\mathrm{s}_{(\theta+\gamma_2)} - (y_{f_2} - y(\theta))\,\mathrm{c}_{(\theta+\gamma_2)}\right] \end{bmatrix} \right)$$

where

$$\begin{aligned} x(\theta) &= k_{0_x} + k_{1_x}\mathrm{s}_\theta + k_{2_x}\mathrm{c}_\theta + k_{3_x}\mathrm{c}_\theta\mathrm{s}_\theta + k_{4_x}\mathrm{s}_\theta^2 \\ y(\theta) &= k_{0_y} + k_{1_y}\mathrm{s}_\theta + k_{2_y}\mathrm{c}_\theta + k_{3_y}\mathrm{c}_\theta\mathrm{s}_\theta + k_{4_y}\mathrm{s}_\theta^2 \end{aligned}$$

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta + k_3\mathrm{c}_\theta\mathrm{s}_\theta + k_4\mathrm{s}_\theta^2 + k_5\mathrm{c}_\theta^2$$

where

$$k_0 = -\tfrac{x_a}{\rho}k_{0_y}\mathrm{s}_{(\gamma_2-\gamma_1)} + \tfrac{y_a}{\rho}k_{0_x}\mathrm{s}_{(\gamma_2-\gamma_1)} + z_a\mathrm{s}_{(\gamma_2-\gamma_1)}$$

$$k_1 = -\tfrac{x_a}{\rho}k_{1_y}\mathrm{s}_{(\gamma_2-\gamma_1)} + \tfrac{y_a}{\rho}k_{1_x}\mathrm{s}_{(\gamma_2-\gamma_1)}$$

$$k_2 = -\tfrac{x_a}{\rho}k_{2_y}\mathrm{s}_{(\gamma_2-\gamma_1)} + \tfrac{y_a}{\rho}k_{2_x}\mathrm{s}_{(\gamma_2-\gamma_1)}$$

$$\begin{aligned} k_3 = {} & \tfrac{x_a}{\rho}\left[(x_{f_2} - x_{f_1})\mathrm{s}_{(\gamma_2+\gamma_1)} - (y_{f_2} - y_{f_1})\mathrm{c}_{(\gamma_2+\gamma_1)} - k_{3_y}\mathrm{s}_{(\gamma_2-\gamma_1)}\right] \\ & + \tfrac{y_a}{\rho}\left[(y_{f_1} - y_{f_2})\mathrm{s}_{(\gamma_2+\gamma_1)} - (x_{f_2} - x_{f_1})\mathrm{c}_{(\gamma_2+\gamma_1)} + k_{3_x}\mathrm{s}_{(\gamma_2-\gamma_1)}\right] \end{aligned}$$

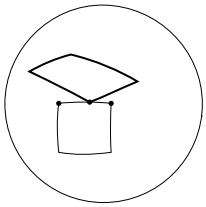
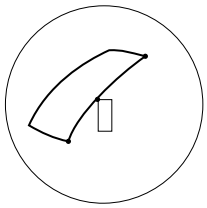
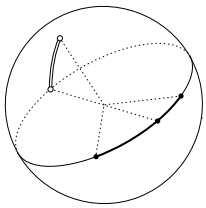
$$\begin{aligned} k_4 = {} & \tfrac{x_a}{\rho}\left[(x_{f_2} - x_{f_1})\mathrm{c}_{\gamma_1}\mathrm{c}_{\gamma_2} + y_{f_2}\mathrm{c}_{\gamma_1}\mathrm{s}_{\gamma_2} - y_{f_1}\mathrm{s}_{\gamma_1}\mathrm{c}_{\gamma_2} - k_{4_y}\mathrm{s}_{(\gamma_2-\gamma_1)}\right] \\ & + \tfrac{y_a}{\rho}\left[(y_{f_2} - y_{f_1})\mathrm{s}_{\gamma_1}\mathrm{s}_{\gamma_2} + x_{f_2}\mathrm{s}_{\gamma_1}\mathrm{c}_{\gamma_2} - x_{f_1}\mathrm{c}_{\gamma_1}\mathrm{s}_{\gamma_2} + k_{4_x}\mathrm{s}_{(\gamma_2-\gamma_1)}\right] \end{aligned}$$

$$\begin{aligned} k_5 = {} & \tfrac{x_a}{\rho}\left[(x_{f_2} - x_{f_1})\mathrm{s}_{\gamma_1}\mathrm{s}_{\gamma_2} - y_{f_2}\mathrm{s}_{\gamma_1}\mathrm{c}_{\gamma_2} + y_{f_1}\mathrm{c}_{\gamma_1}\mathrm{s}_{\gamma_2}\right] \\ & + \tfrac{y_a}{\rho}\left[(y_{f_2} - y_{f_1})\mathrm{c}_{\gamma_1}\mathrm{c}_{\gamma_2} - x_{f_2}\mathrm{c}_{\gamma_1}\mathrm{s}_{\gamma_2} + x_{f_1}\mathrm{s}_{\gamma_1}\mathrm{c}_{\gamma_2}\right] \end{aligned}$$

If $-\mathbf{F}_1$ and $-\mathbf{F}_2$ correspond to a single contact point, then this equation may be simplified, since $x_{f_1} = x_{f_2}$, $y_{f_1} = y_{f_2}$, and $\gamma_2 = \gamma_1 + 2\alpha$.

VE-EV Edges

There are several cases which must be considered, some of which are solved by previously-derived equations. These cases are summarized below:

Critical condition	Coincident force-sphere features	Page
	1. Force-polygon edge and ve-ray	276 (top)
	2. Force-polygon edge and ev-ray	283
	3. Edge formed by two ve-rays and force-polygon vertex	277 (top)
	4. Edge formed by two ev-rays and force-polygon vertex	284
	5. Edge formed by one ve-ray and one ev-ray and force-polygon vertex	285
	6. Edge formed by two ve-rays and a non-negated ev-ray	286
	7. Edge formed by two ev-rays and a non-negated ve-ray	286

We will re-arrange the class-3 obstacle edge equations to a more convenient form:

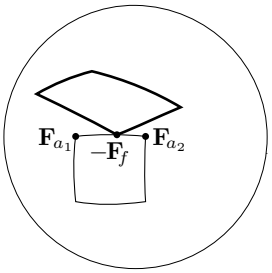
$$x(\theta) - x_f = \frac{K_{0_x} + K_{1_x}\mathrm{s}_\theta + K_{2_x}\mathrm{c}_\theta + K_{3_x}\mathrm{c}_\theta\mathrm{s}_\theta + K_{4_x}\mathrm{s}_\theta^2 + K_{5_x}\mathrm{c}_\theta^2}{k_{6_{xy}}\mathrm{s}_\theta + k_{7_{xy}}\mathrm{c}_\theta}$$

where

$$\begin{aligned}
K_{0_x} &= k_{3_x} \\
K_{1_x} &= k_{4_x} + k_{6_{xy}}(k_{0_x} - x_f) \\
K_{2_x} &= k_{5_x} + k_{7_{xy}}(k_{0_x} - x_f) \\
K_{3_x} &= k_{6_{xy}}k_{2_x} + k_{7_{xy}}k_{1_x} \\
K_{4_x} &= k_{6_{xy}}k_{1_x} \\
K_{5_x} &= k_{7_{xy}}k_{2_x}
\end{aligned}$$

and similarly for $y(\theta) - y_f$, defining terms $K_{0_y} \ldots K_{5_y}$. The $k_{6_{xy}}$ and $k_{7_{xy}}$ terms exploit the fact that the denominator terms are identical for the x and y class-3 obstacle edge equations.

VE-EV Case 2: An ev-ray



$$0 = -\mathbf{F}_f(\theta) \cdot [\mathbf{F}_{a_1} \times \mathbf{F}_{a_2}]$$

$$0 = \begin{bmatrix} \mathrm{c}_{(\theta+\gamma)} \\ \mathrm{s}_{(\theta+\gamma)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma)}\right] \end{bmatrix} \cdot \underbrace{\left(\begin{bmatrix} x_{a_1} \\ y_{a_1} \\ z_{a_1} \end{bmatrix} \times \begin{bmatrix} x_{a_2} \\ y_{a_2} \\ z_{a_2} \end{bmatrix} \right)}_{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}$$

where $x(\theta) - x_f$ and $y(\theta) - y_f$ are applied from page 282.

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta + k_3\mathrm{c}_\theta\mathrm{s}_\theta + k_4\mathrm{s}_\theta^2 + k_5\mathrm{c}_\theta^2 + k_6\mathrm{c}_\theta\mathrm{s}_\theta^2 + k_7\mathrm{c}_\theta^2\mathrm{s}_\theta + k_8\mathrm{s}_\theta^3 + k_9\mathrm{c}_\theta^3$$

where

$$k_0 = 0$$

$$k_1 = -\tfrac{z_A}{\rho}\left(K_{0_x}\mathrm{c}_\gamma + K_{0_y}\mathrm{s}_\gamma\right)$$

$$k_2 = -\tfrac{z_A}{\rho}\left(K_{0_x}\mathrm{s}_\gamma - K_{0_y}\mathrm{c}_\gamma\right)$$

$$k_3 = -\tfrac{z_A}{\rho}\left(K_{1_x}\mathrm{s}_\gamma - K_{1_y}\mathrm{c}_\gamma + K_{2_x}\mathrm{c}_\gamma + K_{2_y}\mathrm{s}_\gamma\right) + y_A k_{7_{xy}}\mathrm{c}_\gamma - x_A k_{7_{xy}}\mathrm{s}_\gamma + y_A k_{6_{xy}}\mathrm{s}_\gamma + x_A k_{6_{xy}}\mathrm{c}_\gamma$$

$$k_4 = -\tfrac{z_A}{\rho}\left(K_{1_x}\mathrm{c}_\gamma + K_{1_y}\mathrm{s}_\gamma\right) + y_A k_{6_{xy}}\mathrm{c}_\gamma - x_A k_{6_{xy}}\mathrm{s}_\gamma$$

$$k_5 = -\tfrac{z_A}{\rho}\left(K_{2_x}\mathrm{s}_\gamma - K_{2_y}\mathrm{c}_\gamma\right) + y_A k_{7_{xy}}\mathrm{s}_\gamma + x_A k_{7_{xy}}\mathrm{c}_\gamma$$

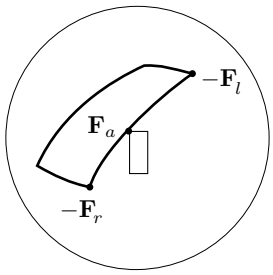
$$k_6 = -\tfrac{z_A}{\rho}\left(k_{7_{xy}}k_{1_x}\mathrm{c}_\gamma + k_{7_{xy}}k_{1_y}\mathrm{s}_\gamma\right)$$

$$k_7 = -\tfrac{z_A}{\rho}\left(k_{6_{xy}}k_{2_x}\mathrm{s}_\gamma - k_{6_{xy}}k_{2_y}\mathrm{c}_\gamma\right)$$

$$k_8 = -\tfrac{z_A}{\rho}\left(K_{4_x}\mathrm{c}_\gamma + K_{4_y}\mathrm{s}_\gamma\right)$$

$$k_9 = -\tfrac{z_A}{\rho}\left(K_{5_x}\mathrm{s}_\gamma - K_{5_y}\mathrm{c}_\gamma\right)$$

VE-EV Case 4: Two ev-rays



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_r(\theta) \times -\mathbf{F}_l(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_{(\theta+\gamma_r)} \\ \mathrm{s}_{(\theta+\gamma_r)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma_r)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma_r)}\right] \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\theta+\gamma_r+2\alpha)} \\ \mathrm{s}_{(\theta+\gamma_r+2\alpha)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma_r+2\alpha)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma_r+2\alpha)}\right] \end{bmatrix} \right)$$

where $\gamma_r + 2\alpha = \gamma_l$, and $x(\theta) - x_f$ and $y(\theta) - y_f$ are applied from page 282.

$$0 = k_0 + k_1 \mathrm{s}_\theta + k_2 \mathrm{c}_\theta + k_3 \mathrm{c}_\theta \mathrm{s}_\theta + k_4 \mathrm{s}_\theta^2 + k_5 \mathrm{c}_\theta^2$$

where

$$k_0 = \tfrac{1}{\rho}(y_a K_{0_x} - x_a K_{0_y})$$

$$k_1 = \tfrac{1}{\rho}(y_a K_{1_x} - x_a K_{1_y}) + z_a k_{6_{xy}}$$

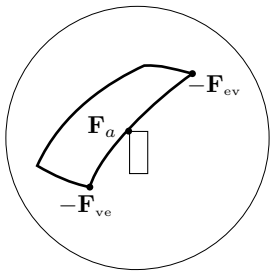
$$k_2 = \tfrac{1}{\rho}(y_a K_{2_x} - x_a K_{2_y}) + z_a k_{7_{xy}}$$

$$k_3 = \tfrac{1}{\rho}(y_a K_{3_x} - x_a K_{3_y})$$

$$k_4 = \tfrac{1}{\rho}(y_a K_{4_x} - x_a K_{4_y})$$

$$k_5 = \tfrac{1}{\rho}(y_a K_{5_x} - x_a K_{5_y})$$

VE-EV Case 5: One ve-ray and one ev-ray



$$0 = \mathbf{F}_a \cdot [-\mathbf{F}_{\text{ve}}(\theta) \times -\mathbf{F}_{\text{ev}}(\theta)]$$

$$0 = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \cdot \left(\begin{bmatrix} \mathrm{c}_\phi \\ \mathrm{s}_\phi \\ -\frac{r}{\rho}\mathrm{s}_{(\theta+\nu-\phi)} \end{bmatrix} \times \begin{bmatrix} \mathrm{c}_{(\theta+\gamma)} \\ \mathrm{s}_{(\theta+\gamma)} \\ \frac{1}{\rho}\left[(x_f - x(\theta))\,\mathrm{s}_{(\theta+\gamma)} - (y_f - y(\theta))\,\mathrm{c}_{(\theta+\gamma)}\right] \end{bmatrix} \right)$$

where $\phi = \phi_{\text{ve}}$, $\gamma = \gamma_{\text{ev}}$, and $x(\theta) - x_f$ and $y(\theta) - y_f$ are applied from page 282.

$$0 = k_0 + k_1\mathrm{s}_\theta + k_2\mathrm{c}_\theta + k_3\mathrm{c}_\theta\mathrm{s}_\theta + k_4\mathrm{s}_\theta^2 + k_5\mathrm{c}_\theta^2 + k_6\mathrm{c}_\theta\mathrm{s}_\theta^2 + k_7\mathrm{c}_\theta^2\mathrm{s}_\theta + k_8\mathrm{s}_\theta^3 + k_9\mathrm{c}_\theta^3$$

where

$$k_0 = 0$$

$$k_1 = \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(-K_{0_y}\mathrm{s}_\gamma - K_{0_x}\mathrm{c}_\gamma)$$

$$k_2 = \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(K_{0_y}\mathrm{c}_\gamma - K_{0_x}\mathrm{s}_\gamma)$$

$$\begin{aligned} k_3 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(K_{1_y}\mathrm{c}_\gamma - K_{1_x}\mathrm{s}_\gamma - K_{2_y}\mathrm{s}_\gamma - K_{2_x}\mathrm{c}_\gamma)) \\ & + z_a[(k_{7_{xy}}\mathrm{c}_\gamma + k_{6_{xy}}\mathrm{s}_\gamma)\mathrm{c}_\phi + (k_{7_{xy}}\mathrm{s}_\gamma - k_{6_{xy}}\mathrm{c}_\gamma)\mathrm{s}_\phi] \end{aligned}$$

$$\begin{aligned} k_4 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(-K_{1_y}\mathrm{s}_\gamma - K_{1_x}\mathrm{c}_\gamma) \\ & + z_a(k_{6_{xy}}\mathrm{c}_\gamma\mathrm{c}_\phi + k_{6_{xy}}\mathrm{s}_\gamma\mathrm{s}_\phi) \end{aligned}$$

$$\begin{aligned} k_5 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(K_{2_y}\mathrm{c}_\gamma - K_{2_x}\mathrm{s}_\gamma) \\ & + z_a(k_{7_{xy}}\mathrm{s}_\gamma\mathrm{c}_\phi - k_{7_{xy}}\mathrm{c}_\gamma\mathrm{s}_\phi) \end{aligned}$$

$$\begin{aligned} k_6 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(-k_{7_{xy}}k_{1_y}\mathrm{s}_\gamma - k_{7_{xy}}k_{1_x}\mathrm{c}_\gamma) \\ & + \tfrac{x_a}{\rho}[k_{6_{xy}}\mathrm{c}_\gamma\mathrm{s}_{(\nu-\phi)} + (k_{7_{xy}}\mathrm{c}_\gamma + k_{6_{xy}}\mathrm{s}_\gamma)\mathrm{c}_{(\nu-\phi)}] \\ & - \tfrac{y_a}{\rho}[-k_{6_{xy}}\mathrm{s}_\gamma\mathrm{s}_{(\nu-\phi)} + (-k_{7_{xy}}\mathrm{s}_\gamma + k_{6_{xy}}\mathrm{c}_\gamma)\mathrm{c}_{(\nu-\phi)}] \end{aligned}$$

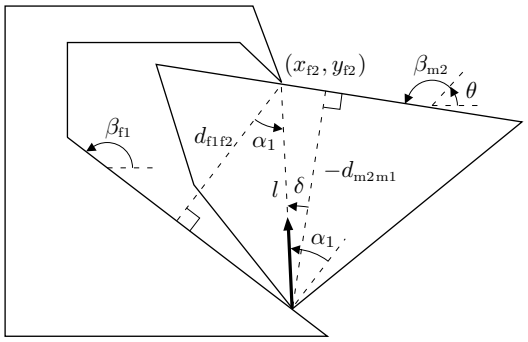
$$\begin{aligned} k_7 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(k_{6_{xy}}k_{2_y}\mathrm{c}_\gamma - k_{6_{xy}}k_{2_x}\mathrm{s}_\gamma) \\ & + \tfrac{x_a}{\rho}[(k_{7_{xy}}\mathrm{c}_\gamma + k_{6_{xy}}\mathrm{s}_\gamma)\mathrm{s}_{(\nu-\phi)} + k_{7_{xy}}\mathrm{s}_\gamma\mathrm{c}_{(\nu-\phi)}] \\ & - \tfrac{y_a}{\rho}[(-k_{7_{xy}}\mathrm{s}_\gamma + k_{6_{xy}}\mathrm{c}_\gamma)\mathrm{s}_{(\nu-\phi)} + k_{7_{xy}}\mathrm{c}_\gamma\mathrm{c}_{(\nu-\phi)}] \end{aligned}$$

$$\begin{aligned} k_8 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(-K_{4_y}\mathrm{s}_\gamma - K_{4_x}\mathrm{c}_\gamma) \\ & + \tfrac{x_a}{\rho}k_{6_{xy}}\mathrm{c}_\gamma\mathrm{c}_{(\nu-\phi)} \\ & + \tfrac{y_a}{\rho}k_{6_{xy}}\mathrm{s}_\gamma\mathrm{c}_{(\nu-\phi)} \end{aligned}$$

$$\begin{aligned} k_9 = {} & \tfrac{1}{\rho}(x_a\mathrm{s}_\phi - y_a\mathrm{c}_\phi)(K_{5_y}\mathrm{c}_\gamma - K_{5_x}\mathrm{s}_\gamma) \\ & + \tfrac{x_a}{\rho}k_{7_{xy}}\mathrm{s}_\gamma\mathrm{s}_{(\nu-\phi)} \\ & - \tfrac{y_a}{\rho}k_{7_{xy}}\mathrm{c}_\gamma\mathrm{s}_{(\nu-\phi)} \end{aligned}$$

To identify force-closure critical θ-values on ve-ev obstacle edges, critical-value equations may be derived which have θ-roots corresponding to hemispherical critical conditions on the force sphere. These equations were developed using the dot-product analysis applied previously, but the resulting analytic equations exhibited poor numerical robustness properties during testing. This section presents a simpler technique for identifying these critical values that is both more robust and more efficient. The technique exploits the fact that the transition between a force-closure and non-force-closure condition can only occur for a ve-ev contact when the ray of one friction cone passes through the base of the opposite friction cone. This condition is equivalent to the hemispherical force-sphere condition used in the ve-ve derivation shown at the bottom of page 279. The derivations shown below illustrate this construction for the left rays of the ve and ev contact friction cones; the constructions for the right rays are analogous.

VE-EV Case 6: What angle will cause $\mathbf{F}_{\mathrm{ve}_l}$ to pass through $(x_{\mathrm{f2}}, y_{\mathrm{f2}})$?



$$\begin{aligned} l\cos(\alpha_1) &= d_{\mathrm{f1f2}} \\ l &= \frac{d_{\mathrm{f1f2}}}{\cos(\alpha_1)} \end{aligned}$$

$$\begin{aligned} l\cos(\delta) &= -d_{\mathrm{m2m1}} \\ \delta &= \arccos\left(\frac{-d_{\mathrm{m2m1}}}{l}\right) \end{aligned}$$

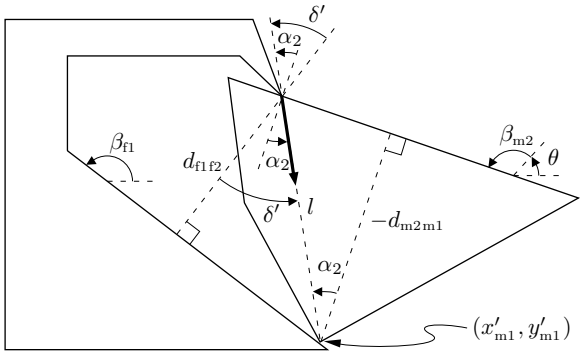
By inspection:

$$\beta_{\mathrm{f1}} - \tfrac{\pi}{2} + \alpha_1 - \delta = \theta + \beta_{\mathrm{m2}} - \tfrac{\pi}{2}$$

Substituting and rearranging gives:

$$\boxed{\theta = \beta_{\mathrm{f1}} - \beta_{\mathrm{m2}} + \alpha_1 - \arccos\left(\tfrac{-d_{\mathrm{m2m1}}}{d_{\mathrm{f1f2}}}\cos(\alpha_1)\right)}$$

VE-EV Case 7: What angle will cause $\mathbf{F}_{\mathrm{ev}_l}$ to pass through $(x'_{\mathrm{m1}}, y'_{\mathrm{m1}})$?



$$\begin{aligned} l\cos(\alpha_2) &= -d_{\mathrm{m2m1}} \\ l &= \frac{-d_{\mathrm{m2m1}}}{\cos(\alpha_2)} \end{aligned}$$

$$\begin{aligned} l\cos(\delta') &= d_{\mathrm{f1f2}} \\ \delta' &= \arccos\left(\frac{d_{\mathrm{f1f2}}}{l}\right) \end{aligned}$$

By inspection:

$$\delta' = \theta + \beta_{\mathrm{m2}} - \tfrac{\pi}{2} + \alpha_2 - (\beta_{\mathrm{f1}} - \tfrac{\pi}{2})$$

Substituting and rearranging gives:

$$\boxed{\theta = \beta_{\mathrm{f1}} - \beta_{\mathrm{m2}} - \alpha_2 + \arccos\left(\tfrac{d_{\mathrm{f1f2}}}{-d_{\mathrm{m2m1}}}\cos(\alpha_2)\right)}$$

Appendix 5: The *CO* Algorithm

This appendix elaborates on the brief discussion of the *CO* algorithm presented earlier. In particular, the following sections define the notion of globally-convex contact features, discuss the nuances of incrementally constructing configuration obstacles, and present a pseudo-code summary of the *CO* algorithm.

5.1. Global Convexity

For some facets, non-local conflicts are impossible because both contact features are *globally convex.* A polygon edge is globally convex if all other points of the polygon are strictly inside the half-space formed by the line containing the edge. A polygon vertex is globally convex if both edges adjacent to the vertex are globally convex (Figure 185).

If the contact edge and vertex for a facet are both globally convex, then non-local conflicts are impossible for the facet. To see why, consider the example ve contact condition shown in Figure 186. The contact edge and vertex are both globally convex; the associated half-space constraints are shown shaded. If the moving-polygon is swept through the range of locally-consistent orientations, the two shaded regions never intersect. Since all points of the moving-polygon and fixed-polygon are contained within these regions, this implies that no features of the two polygons can ever touch, except for the features local to the ve contact. Thus, non-local conflicts cannot occur for this ve contact. A similar argument applies to ev contacts.

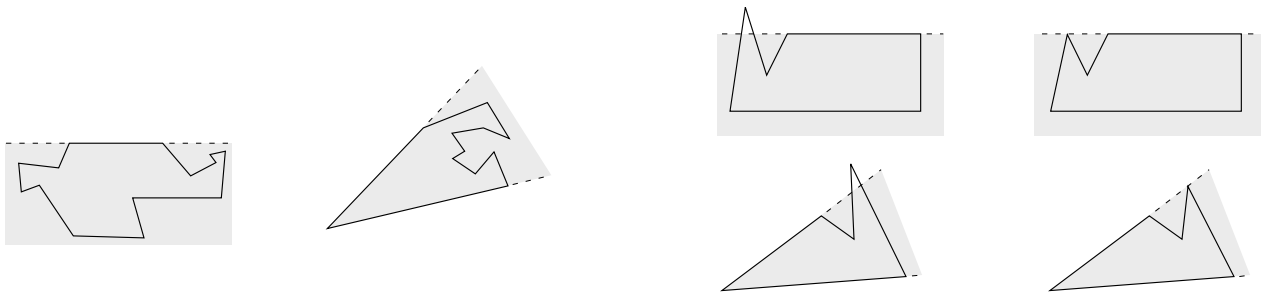


Figure 185: (a) A globally-convex edge. (b) A globally-convex vertex. (c) Examples that are not globally convex; the cases on the right emphasize the requirement that non-local points must be strictly contained within the appropriate half-spaces.

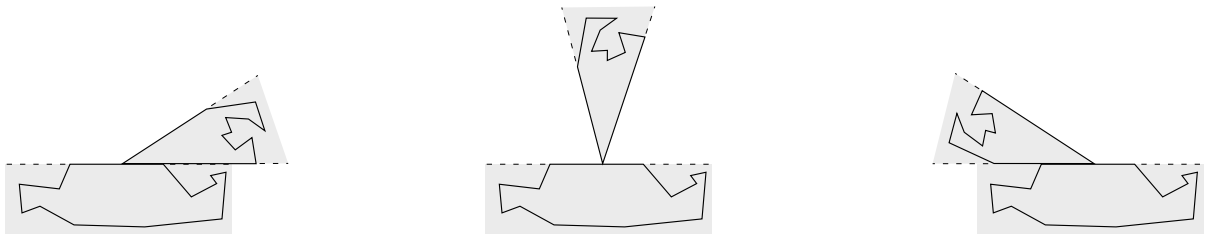


Figure 186: A contact between two globally-convex features.

5.2. Incremental Surface Construction

The *CO* algorithm is capable of constructing individual facets of the configuration obtacle on demand, and in any order. Thus a planner can request a specific obstacle feature, and the algorithm can produce the facets in the neighborhood of the requested feature, or determine that the feature does not exist on the obstacle surface. For example, if a planner is given the goal of achieving an interval of configurations satisfying the contact condition v_2e_6-v_5e_1, the planner can request the corresponding obstacle edge, and the *CO* algorithm will construct the v_2e_6 and v_5e_1 adjacent facets. When the algorithm returns, the requested edge and its surrounding facets will be represented in the returned c-obstacle data structure; the absence of the edge will imply that it does not appear on the obstacle surface. If the planning algorithm later requires additional facets adjacent to the v_2e_6 and v_5e_1 facets, the *CO* algorithm can construct these as required.

The core of the *CO* algorithm is the *complete-facet* procedure, which constructs a facet on the obstacle surface. This procedure performs all of the steps of the previous *simplified-CO* algorithm: The local edges of the facet are created, the facet is intersected with other facets to form conflict edges, superimposed edges on the facet are merged to form mc0-edges, extraneous edges are removed, and unreachable connected components of the facet are discarded. Executing this procedure for each possible obstacle facet ultimately produces a data structure identical to the data structure produced by the *simplified-CO* algorithm.

However, the *complete-facet* procedure is not simply the result of applying the construction steps of the *simplified-CO* algorithm on an individual facet. In addition to these basic construction steps, the *complete-facet* procedure incorporates side-effects that assure that information obtained during early facet constructions is properly utilized by later constructions. These side-effects are best understood by considering the invariants maintained by the *simplified-CO* algorithm:

- All possible facets, edges, and vertices are created.
- All topological neighbors are properly identified.
- No reachable edges are ever removed.
- All unreachable edges are eventually removed.

The side-effects employed by the *complete-facet* procedure are designed to maintain these invariants when the obstacle surface is constructed incrementally. The first two invariants address the construction of possible features, and the last two invariants address the deletion of unreachable features.

The last two invariants are easy to satisfy. The deletion steps used in the *simplified-CO* algorithm are already structured to process facets individually, so the *complete-facet* procedure uses these steps without significant modification. Assuming that all features on a given facet have been created before these deletion steps are executed, then these steps remove all unreachable features from the facet, and retain all reachable features. As long as no features are added to the facet later, the facet will be properly represented indefinitely.

The first two invariants are more subtle. They require that the algorithm construct all possible features, and correctly identify their topological adjacency relationships. Further, the discussion of the previous paragraph places temporal constraints on these invariants: All of the features that can possibly appear on a facet must be constructed before the deletion steps are encountered, and no features on the facet should be constructed after the deletion steps are executed. Since the order of facet construction may be arbitrarily chosen by the calling program, these invariants must be enforced by information placed in the data structure that allows communication between subsequent executions of the *complete-facet* procedure.

The side-effects of the *complete-facet* procedure produce and maintain this information. These side-effects insure that all features that can arise on a facet have been constructed by the time the deletion steps are executed, and further insure that no features are added to the facet once these deletion steps are complete. The side-effects primarily affect the construction of obstacle edges, of which there are three basic categories: local, conflict, and multiple-class-0. We will review the issues associated with each of these categories, and the corresponding side-effects employed.

5.2.1. Local Edges

The edges constructed for a given vv or ee contact may border up to four facets. Thus, when the local analysis is performed for a given facet F, local edges are added to the edges field of several facets other than F. This analysis should not be repeated, for several reasons. In addition to obvious efficiency considerations, topological errors can result. For example, suppose that while constructing the facet F_1, the vv edges for the minimum p-value constraint are constructed, and they border four facets: $\{F_1, F_2, F_3, F_4\}$. The local analysis will construct these edges and place them in the edges field of each facet. Further, suppose that F_1 is only adjacent to F_2 and F_3; F_1 and F_4 share no common border. Now suppose that non-local interactions make the vv edges unreachable on F_1, so they are deleted from F_1, F_2, and F_3. Now if F_2 is constructed next, its edges field will not contain the vv edges previously constructed, so it will also perform a local analysis to construct these edges. This local analysis will add edges to all four facets again, causing two topological errors: (1) Edges are being added to F_1 after its deletion steps have been executed, and (2) F_4 now contains two copies of the vv edge. This is clearly an undesirable situation. The *CO* algorithm avoids these problems by including a local-contacts-constructed field in each c-facet, indicating the relevant local analysis steps that have been previously completed.

5.2.2. Conflict Edges

The non-local analysis procedures require the intersection of all pairs of facets to identify conflict edges. To avoid problems analogous to the local-analysis problems described described above, no pair of facets should be intersected more than once. The *CO* algorithm insures this by maintaining a set containing all of the incomplete facets. This incomplete-facet-set is initialized to contain every possible facet, and facets are removed from the set as they are completed. When the *complete-facet* procedure executes its non-local conflict analysis for a facet F, it removes F from the incomplete-facet-set, and then intersects F with every facet remaining in the incomplete-facet-set. Since previous facet construction steps included F in their intersection steps, this assures that F has been intersected with every other obstacle facet. This implies that all conflict edges that can arise on F have been constructed. Further, since F no longer appears in the incomplete facet-set, later facet constructions will not include F in their facet-intersection step, and additional conflict edges will be never be added to F.

5.2.3. Multiple-Class-0 Edges

The non-generic multiple-class-0 edges produce the most subtle interactions between executions of the *complete-facet* procedure. To see why, consider the example shown in Figure 187. Suppose that the first facet constructed is facet v_1e_9. The non-local conflict analysis will produce class-0 conflict edges for the contacts v_1e_9-v_4e_5 and v_1e_9-v_7e_1. These class-0 conflict edges will be merged to form the multiple-class-0 edge with the contacts v_1e_9-v_4e_5-v_7e_1, with left and right adjacent facets v_1e_9 and v_7e_1. Thus, the resulting multiple-class-0 edge will not appear in the edges field of the facet v_4e_5. Now suppose that the facet v_4e_5 is completed next. Since the v_1e_9 facet has already been completed, it will not be intersected with the v_4e_5 facet, but the incomplete facet v_7e_1 will be intersected. This will produce a class-0 conflict edge with the contacts v_4e_5-v_7e_1. Since there are no other class-0 conflict-edges present in the v_4e_5 facet, this edge is not merged, and remains present with its left and right facet neighbors v_4e_5 and v_7e_1. This topological error can lead to a variety of subsequent problems, depending on the geometry of the contact features.

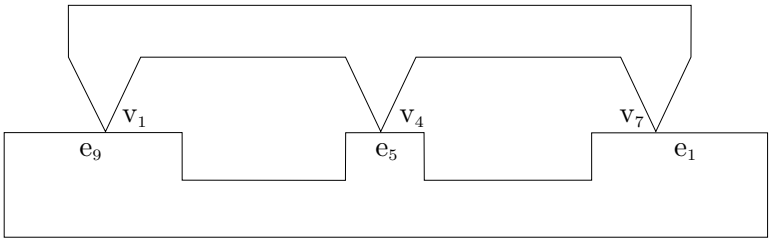


Figure 187: A multiple-class-0 contact condition.

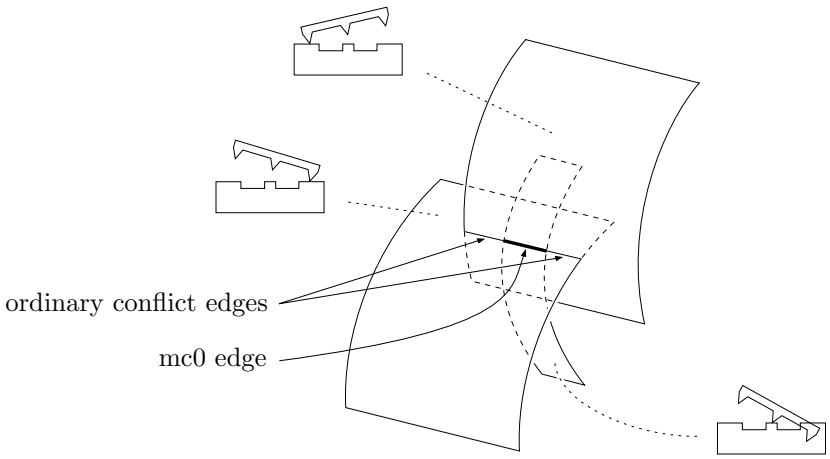


Figure 188: Facets intersecting to form a multiple-class-0 edge.

Problems such as this can be remedied by including appropriate side-effects; these side-effects are motivated by an understanding of the structure of multiple-class-0 edges. Figure 188 shows an example mc0-edge, and the locally-consistent facets that intersect to form it. Several observations become apparent:

1. The endpoints of the merged edge are defined by the innermost limits imposed by the locally-consistent facets. Regardless of the order that the facets are considered, the resulting merged endpoints will be the same.

2. All facets participate in the merging process the first time the merged edge is constructed. Regardless of which facet is constructed first, all of the other facets will be intersected with the facet, producing conflict edges. The result of merging all of these segments will give rise to the illustrated merged edge, and will include all of the intersecting facets.

3. This merged edge will never be changed during later mc0-edge analysis steps. This is because all of the non-generically intersecting facets that contain this edge were included in the first merging operation. Therefore while later facet-construction operations may give rise to other conflict edges that will be superimposed with the mc0-edge (as in the example described above), the merged result of including the new conflict edges will not change the extent or facet-neighbors of the mc0-edge. This implies that once we construct an mc0-edge and its adjacent facets, we will never need to modify its neighbor fields, or split it during an mc0-analysis step.

These observations allow us to formulate the following strategy for assuring the correctness of mc0-edges constructed by the *complete-facet* procedure: When an mc0-edge is constructed for the first time, we will set its endpoints and neighbor fields, and also a flag indicating that this analysis does not need to be repeated. We will also arrange the merging process to assure that when an mc0-edge is superimposed with a conflict edge, the mc0-edge is preserved. Finally, we will include the mc0-edge in the edges field of all facets contributing to it, so that later mc0-edge analysis steps will be properly executed. Some care is required to assure the integrity of this side-effect as other facet-construction operations are performed; we will elaborate on this issue below. These side-effects will support the following invariants:

1. All mc0-edge constructions are proper.

2. All reachable mc0-edges persist indefinitely.

3. All unreachable mc0-edges are deleted forever.

We will review each of these invariants below.

1. All mc0-edge constructions are proper. We assure that all mc0-edge constructions are proper by including mc0-edges in the edges field of their contributing facets. This allows conflict edges that arise later to be merged with the mc0-edge; the superimposed portion of these new conflict edges will be subsumed by the previously-constructed mc0-edge, which has correct facet-neighbors. Thus the presence of the original mc0-edge assures correct handling of subsequent superimposed conflict edges.

There are some subtleties associated with this strategy. For example, the original mc0-edge may be split by an intersecting non-class-0 edge; if this occurs, the mc0-edge will now be represented by two c-edge data records — the original mc0-edge data record (now shortened), and a clone of the original mc0-edge data record (comprising the remaining edge segment). To assure proper handling of future conflict edges, this new c-edge must be added to the partially-constructed facets that contributed to the mc0-edge. This is facilitated by including an mc0-partial-facets field in the c-edge data record, and modifying the *split-edge* procedure to add the cloned c-edge data record it produces to all facets in the mc0-partial-facets field.

An additional subtlety arises when mc0-edges are deleted because they are unreachable. To assure proper handling of future conflict edges, a deleted mc0-edge must be preserved even though it is unreachable. It is not sufficient to simply remove the deleted edge from the edges field of its adjacent vertices and facets, because the $\text{vertex}_{\min}$ and $\text{vertex}_{\max}$ pointers defining the endpoints of the deleted edge will still point to these c-vertex data records. If the coordinates of either vertex are changed for some reason, then the metric properties of the deleted mc0-edge will be corrupted, leading to erroneous mc0-analysis steps later.

To prevent this, we will discard unreachable mc0-edges by topologically isolating them. This is accomplished by removing an unreachable mc0-edge from the edges field of its vertices and adjacent complete facets, and setting its $\text{vertex}_{\min}$ and $\text{vertex}_{\max}$ fields to new c-vertex data records with the same coordinates as the original edge endpoints. This isolation procedure will assure that the mc0-edge has intact metric properties when subsequent *complete-facet* procedures use it to detect superimposed conflict edges.

2. All reachable mc0-edges persist indefinitely. Once an mc0-edge is constructed and determined to be reachable, it will be retained in the edges field of its neighboring facets. As the computation progresses, the mc0-edge must not be removed from these facets. This requires careful implementation of the merge procedure to assure that when an mc0-edge and a conflict edge are merged, the mc0-edge is preserved. From the previous discussion, we know that a fully-constructed mc0-edge will never be split by superimposed conflict edges constructed later; this allows us to write a merging procedure that always preserves mc0-edges. This procedure is described in the pseudo-code that follows.

3. All unreachable mc0-edges are deleted forever. If an mc0-edge is deleted from a neighboring facet because it is unreachable, it must never be re-inserted into the facet's edges field. Because the facet-neighbors of an mc0-edge are only analyzed and set once when the mc0-edge is first constructed, this error will never occur, even though the mc0-edge may be merged with conflict edges after it has been deleted. This places another special requirement on the merge procedure: if an isolated mc0-edge is merged with a conflict edge that is longer than the mc0-edge, the isolated mc0-edge should not be topologically connected to the remaining conflict-edge segment. The reason for this is that if the leftover conflict-edge segment is reachable, then the mc0-edge may remain attached to the obstacle data structure, even though it is unreachable. An isolated? field is included in the c-edge data structure to support this requirement.

5.2.4. Additional Remarks

The previous discussion has explained how the *CO* algorithm uses side-effects to assure correctness as the configuration-obstacle surface is constructed incrementally. There is an additional subtlety imposed by this control structure, which is that special handling is required to construct the contact sets of certain obstacle features. Specifically, the contact-set of obstacle vertices must be obtained through lazy-evaluation, and constructing the complete contact-set of multiple-class-0 edges requires a search for implicit ee contacts.

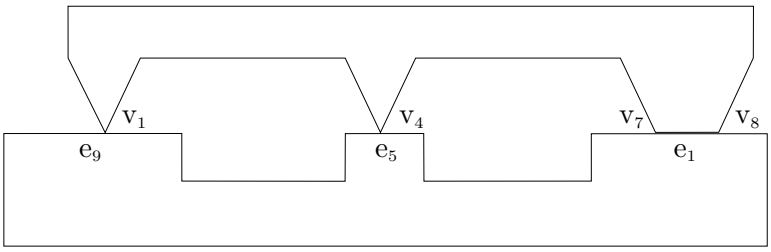


Figure 189: A contact condition that may require identification of implicit ee contacts.

Lazy-evaluation of vertex contact sets. The contact-set of an obstacle vertex may be computed by forming the union of the contact sets of all the edges that intersect at the vertex. This computation is simple to perform once all of the intersecting edges are identified, but will produce erroneous results if some intersecting edges are absent. Therefore this computation must be delayed until all intersecting edges are constructed; this will be the case once all facets adjacent to the vertex have been completed.

Since a partially-constructed obstacle generally contains edges that border a facet that is not yet complete, the endpoints of these edges correspond to vertices that have at least one incomplete facet. Therefore a partially-constructed obstacle data structure contains some vertices whose contact sets cannot be computed using the simple algorithm described above.

The *CO* algorithm solves this problem by using lazy-evaluation to compute c-vertex contact sets. As c-vertex data records are created and manipulated during obstacle construction, their contact-set fields are left blank. When a calling program requests the contact-set of a c-vertex V, the algorithm checks the left and right facets of each edge in the edges field of V, and calls the *complete-facet* procedure for any facets that are not yet complete. This process is repeated until the left and right facets of all edges in V are complete; this assures that all edges intersecting V have been constructed. V's contact set is then constructed by forming the union of the edge contact sets; this result is cached in V's contact-set field for future reference.

Searching for implicit ee contacts in mc0-edges. The *construct-mc0-edges* procedure correctly constructs the endpoints and facet-neighbors of mc0-edges, but may omit ee contacts from an mc0-edge contact-set. See Figure 189. If the v_1e_9 facet is constructed first, the conflict edges v_1e_9-v_4e_5, v_1e_9-v_7e_1, and v_1e_9-v_8e_1 will be placed in the edges field of the v_1e_9 facet, and the mc0-analysis will merge these edges to form an mc0-edge with the contact-set v_1e_9-v_4e_5-v_7e_1-v_8e_1. The contact e_7e_1 is implicit in this contact-set, since v_7 and v_8 are the endpoints of e_7, and both vertices are in contact with e_1. However, the e_7e_1 contact is not explicitly represented, because the v_7e_1 and v_8e_1 facets were not completed before the mc0-edge was constructed.

Some physical analysis programs may require ee contacts to be represented explicitly, so the *CO* algorithm identifies these contacts by detecting topological situations where an ee contact is implicit. Lazy-evaluation is used to defer this construction until it is requested by the calling program.

5.3. Pseudo-Code Summary of the *CO* Algorithm

The following pages present a pseudo-code summary of the *CO* algorithm, and its associated data structures. This algorithm is an extended version of the *simplified-CO* algorithm presented in the Kinematics chapter of Part II, and includes the optimizations described previously. The primary purpose of this pseudo-code is to clarify the optimizations that distinguish the *CO* algorithm from the *simplified-CO* algorithm.

CO $(\mathcal{P}_m, \mathcal{P}_f)$

Initialize the obstacle.
Construct the c-obstacle data record O, setting the moving-polygon and fixed-polygon fields to $\mathcal{P}_m$ and $\mathcal{P}_f$. Set the facet-array field to an empty array of the appropriate size, and set the incomplete-facet-set-constructed? field to false.

Build the facets.
Use the features of $\mathcal{P}_m$ and $\mathcal{P}_f$ to formulate the set $\mathcal{C}$ of all possible ve and ev contact-sets.
For every contact-set $C \in \mathcal{C}$:

Lookup the c-facet.
Fetch the contents of the facet-array slot corresponding to C. The slot may be empty, or may contain a set $\{F\}$, where F is a c-facet that is not yet complete. If the slot is empty, create a new c-facet F by calling the procedure *new-facet*(C, O).

Complete the c-facet.
Call the routine *complete-facet*(F); when this routine returns, the facet-arrray slot for F will contain a set of completed c-facets $\mathcal{F}$ representing all of the reachable configurations for the contact condition C. Further, this procedure has the side-effect of initializing the incomplete-facet-set the first time it is called for a contact condition that is not globally-convex. Under all circumstances, F will not appear in the incomplete-facet-set when this routine exits.

endfor

new-facet (C, O)

Create a c-facet F.

Update O's facet-array to contain the set $\{F\}$ in the slot corresponding to C.

Set the obstacle field to O.

Set the contact-set to C, and fill in the *ve-or-ev*, moving-feature, and fixed-feature fields.

If the contact vertex is not strictly convex, then

Set the stage field to `impossible`, and return F.

else

Set the stage field to `partial`.

Set the $\theta_{\min}$ and $\theta_{\max}$ fields, using the local facet geometry.

If the contact features are globally convex, then

Set the conflict-possible? field to false.

else

Set the conflict-possible? field to true.

Add F to O's incomplete-facet-set, and set F's incomplete-facet-set-entry field to the resulting set entry data object.

endif

Return F.

endif

complete-facet (F)

Determine whether F is locally possible.
If F's stage field is `impossible`, then

> **F is locally impossible; return $\{F\}$.**
> The routine *new-facet* determined that F is impossible because its contact vertex is non-convex. Return $\{F\}$.

else

> **Perform local analysis.**
> Call the procedure *local-facet*(F), which constructs all of the vv and ee edges that can arise on F, and topologically connects these edges at their common endpoints. This procedure also constructs the metric functions describing F's c-surface.
>
> This procedure uses the local-contacts-constructed field to avoid repeating analysis that has previously been completed, and updates this field in other facets to similarly avoid future recomputation. The procedure may access facets have not yet been initialized; the *new-facet* procedure is used to create these facets as necessary.
>
> **Determine whether non-local analysis is necessary.**
> If F's conflict-possible? field is false, then
>
> > **Non-local analysis is not required; return $\{F\}$.**
> > Conflicts are not possible for this facet because the contact vertex and edge are both globally convex. Non-local analysis is not required, so set F's stage field to `complete` and return $\{F\}$.
>
> else
>
> > **Perform non-local analysis.**
> > Call the procedure *non-local-facet*(F), which evaluates non-local interactions to construct a set of c-facets $\mathcal{F}_{\text{reachable}}$, where each c-facet in $\mathcal{F}_{\text{reachable}}$ represents a connected patch of F containing reachable configurations.
> > When this procedure is entered, previous computations may have generated some of the conflict and mc0-edges that can arise on F, and some of these edges may already be deleted. The procedure uses the incomplete-facet-set to identify the remaining facets that must be paired with F to generate conflict edges, and removes F from this set. This procedure initializes the incomplete-facet-set if necessary. Other special steps are included to correctly handle mc0-edges.
> >
> > **Determine whether F contains any reachable configurations.**
> > If $\mathcal{F}_{\text{reachable}} = \emptyset$, then
> >
> > > **F is unreachable; return $\{F\}$.**
> > > Set the stage field of F to `impossible`, and return $\{F\}$.
> >
> > else
> >
> > > **F is reachable; return $\mathcal{F}_{\text{reachable}}$.**
> > > Place $\mathcal{F}_{\text{reachable}}$ in the facet-array slot for F, set the stage of each $F' \in \mathcal{F}_{\text{reachable}}$ to `complete`, and return $\mathcal{F}_{\text{reachable}}$.
> >
> > endif
>
> endif

endif

local-facet (F)

Construct metric information.
Construct the functions $f_x(p,\theta)$, $f_y(p,\theta)$, and $f_p(x,y,\theta)$ that describe the infinite c-surface containing F, and place these in F's metric information fields.

Enumerate local contacts.
Identify the vv and ee contact conditions $\mathcal{C}_{\text{local}}$ that may border F. These are identified from the polygon features adjacent to the contact features; for example, if F's contact-set is $\mathrm{v}_2\mathrm{e}_6$, then F may be bordered by $\mathrm{v}_2\mathrm{v}_6$, $\mathrm{v}_2\mathrm{v}_7$, $\mathrm{e}_1\mathrm{e}_6$, and $\mathrm{e}_2\mathrm{e}_6$ local edges.

Construct edges for local contacts not yet analyzed.
F's local-contacts-constructed field contains the set of contact conditions $\mathcal{C}_{\text{already}} \subseteq \mathcal{C}_{\text{local}}$ that have already been analyzed. These contact conditions should not be analyzed again.
For all contact conditions $C \in \mathcal{C}_{\text{local}} - \mathcal{C}_{\text{already}}$:

Notify other facets that C has been analyzed.
Identify the set of facets $\mathcal{F}$ that may be bordered by an edge with the contact-set C. To prevent future repetition of this analysis, $\mathcal{F}$ should contain all facets that would generate C as a possibly-bordering local contact condition. Add C to the local-contacts-constructed field of each $F' \in \mathcal{F}$. If a c-facet data record does not exist for a given F', create one using the *new-facet* procedure.

Construct local edges for C.
Construct the string of local edges $\mathcal{E}_C$ for the contact condition C, setting the contact-set and metric information fields appropriately. Topologically connect these edges at their common endpoints. Then, for each edge $E_C \in \mathcal{E}_C$:

Set the edge neighbors.
Set the $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields of E_C to the left and right neighboring facets F_{left} and F_{right}, and add E_C to the edges field of F_{left} and F_{right}.

endfor

endfor

Topologically connect vv and ee edges.
Connect the vv and ee edges in F at their common endpoints, which can only occur at the four corners of F's locally-consistent region. Since the edges of a given vv or ee contact-set were connected earlier, this step assures that all local edges in F are topologically connected.

Return.
At this point, all of the vv and ee contacts in $\mathcal{C}_{\text{local}}$ have been analyzed, and the resulting edges that border F have been placed in F's edges field. Some edges may no longer appear in this field, because they were both created and deleted during a previous facet construction step. This is not a problem, since only unreachable edges will be absent.

non-local-facet (F)

Assure that the incomplete-facet-set has been constructed.
Lookup O from the obstacle field of F, and check O's incomplete-facet-set-constructed? field. If this field is set to true, proceed to the next step. Otherwise, this is the first time that this routine has been called, and the facet-array and incomplete-facet-set have not been completely filled in. To construct the missing entries, use the features of $\mathcal{P}_m$ and $\mathcal{P}_f$ to identify the set $\mathcal{C}$ of facet contacts. Then for each contact-set $C \in \mathcal{C}$, lookup the corresponding facet-array slot, and call *new-facet*(C, O) if the slot is empty. This will simultaneously fill the facet-array and incomplete-facet-set as needed. Finally, set O's incomplete-facet-set-constructed? field to true.

Construct conflict edges.
Use F's incomplete-facet-set-entry to remove F from the incomplete-facet-set. Then for every facet F_i remaining in the incomplete-facet-set, call the procedure *construct-conflict-edges*(F, F_i). This constructs the conflict edges that result from intersecting F with each remaining incomplete facet; conflict edges due to complete facets have already been constructed, and deleted if unreachable. Thus when this step is complete, all conflict edges involving F have been constructed, but some unreachable edges may not be present in F's edges field.

Construct mc0-edges.
Call the procedure *construct-mc0-edges*(F), which merges superimposed class-0 edges in F to form multiple-class-0 edges. This procedure produces side-effects to insure that subsequent constructions of multiple-class-0 edges will be performed properly.

Connect edge intersection points.
Assure that all intersecting edges on F are topologically connected; split edges as necessary to produce this connectivity. When splitting, update mc0-partial-facets as required.

Remove extraneous edges.
Use the local topology of F's vertices to identify and remove extraneous edges bordering F. For each extraneous edge E, remove the edge by calling the procedure *isolate-edge*(E, F). When this step is complete, the remaining edges form a collection of closed loops bounding F.

Remove completely buried components.
After extraneous edges are removed, F may still contain connected sets of unreachable configurations. Remove unreachable components of F as follows:

> **Assemble edges into loops.**
> Assemble F's remaining edges into closed loops. Some of these loops will be outermost bounding loops, and others will be nested loops forming holes and islands.
>
> **Split F into connected components.**
> If the loops constructed in the previous step define two or more disconnected components, split F into a set of c-facets $\mathcal{F}$, where each $F' \in \mathcal{F}$ defines a single component of F that is connected without boundary. Otherwise, set $\mathcal{F} = \{F\}$.
>
> **Discard buried components.**
> Filter $\mathcal{F}$ to produce $\mathcal{F}_{\text{reachable}}$, the set of reachable components of F. Determine the reachability of each $F' \in \mathcal{F}$ by choosing a point strictly inside F' and checking the reachability of the corresponding configuration. If F' is reachable, add it to $\mathcal{F}_{\text{reachable}}$; if F' is not reachable, discard each edge E in its edges field by calling the procedure *isolate-edge*(E, F').

Return.
The set $\mathcal{F}_{\text{reachable}}$ contains a collection of c-facets representing the reachable components of F; if F is completely unreachable, then $\mathcal{F}_{\text{reachable}} = \emptyset$. Return $\mathcal{F}_{\text{reachable}}$.

construct-conflict-edges (F_1, F_2)

Check for obvious non-intersections.
Apply simple tests to detect situations where the facets cannot intersect. These occur when the facet θ-intervals are disjoint, or when the geometry of the contact features precludes the corresponding multiple-contact in any configuration. For example, if F_1 and F_2 are both ve facets, they can only intersect when $l_{\mathrm{m1m2}} \in [l_{\mathrm{f1f2_{min}}}, l_{\mathrm{f1f2_{max}}}]$, where l_{m1m2} is the distance between the contact vertices, and $l_{\mathrm{f1f2_{min}}}$ and $l_{\mathrm{f1f2_{max}}}$ are the minimum and maximum distances between points on the contact edges. The function *conflict-possible?*(F_1, F_2) detects these conditions.
If $[\theta_{\mathrm{min}_1}, \theta_{\mathrm{max}_1}] \cap [\theta_{\mathrm{min}_2}, \theta_{\mathrm{max}_2}] = \emptyset$ or *conflict-possible?*$(F_1, F_2) = \mathrm{false}$, then return.

Intersect the c-surfaces.
Construct the set of unbounded curves $\mathcal{C}$ corresponding to the intersection of the F_1 and F_2 c-surfaces, using the kinematic equations given in Appendix . If $\mathcal{C} = \emptyset$, then return.

Analyze the resulting curves.
For each unbounded curve $C \in \mathcal{C}$:

Intersect the curve with F_1.
Use the local geometry of the F_1 contact features to identify a set of intervals $\mathcal{I}_1$ that delimit the curve's intersection with F_1. $\mathcal{I}_1$ is a discrete collection of intervals $I_{1_i} = [p_{\mathrm{min}1_i}, p_{\mathrm{max}1_i}]$ or $I_{1_i} = [\theta_{\mathrm{min}1_i}, \theta_{\mathrm{max}1_i}]$, depending on whether the curve is class-0 or non-class-0. For every p or θ chosen from an I_{1_i} interval, the corresponding point on the curve lies on or within the local constraints of F_1. If $\mathcal{I}_1 = \emptyset$, then return.

Intersect the curve with F_2.
Similarly, use the local geometry of the F_2 contact features to identify a set of intervals $\mathcal{I}_2$ that delimit the curve's intersection with F_2. If $\mathcal{I}_2 = \emptyset$, then return.

Intersect the previous two results.
Intersect the intervals in $\mathcal{I}_1$ and $\mathcal{I}_2$ to construct $\mathcal{I}$, the set of intervals delimiting curve segments that simultaneously satisfy the local constraints of both F_1 and F_2. If $\mathcal{I} = \emptyset$, then return.

Construct c-edges.
For every interval $I \in \mathcal{I}$:

Create a c-edge data record E.

Set the contact set.
Set E's contact-set to the union of the F_1 and F_2 contact-sets.

Set the metric information.
Set E's metric information fields to the functions describing the unbounded curve C.

Set the endpoints.
Use the minimum value in I and the curve functions to compute the point $(x_{\mathrm{min}}, y_{\mathrm{min}}, \theta_{\mathrm{min}})$ on C; construct a c-vertex at this point, and set E's $\mathrm{vertex_{min}}$ field to this c-vertex. Similarly, use the maximum value of I to construct a c-vertex at $(x_{\mathrm{max}}, y_{\mathrm{max}}, \theta_{\mathrm{max}})$, and set E's $\mathrm{vertex_{max}}$ field. Add E to the edges field of both c-vertex records.

Set the facet neighbors.
Set E's $\mathrm{facet_{left}}$ and $\mathrm{facet_{right}}$ fields to F_1 and F_2, using hand-coded decision procedures that consider the contact geometry to identify the left/right ordering. Add E to the edges field of F_1 and F_2.

endfor

endfor

construct-mc0-edges (F)

Form classes of collinear edges.
Assemble all of the class-0 edges in F, and partition them into classes $\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$ such that all edges within a given class $\mathcal{E}_i$ have the same θ-value. Since all edges lie on the facet surface, edges within each class are collinear.

Construct mc0-edges within each class.
For each class $\mathcal{E}_i$:

Topologically merge superimposed edges.
Call the procedure *merge-collinear-class-0-edges*($\mathcal{E}_i$). When this routine returns, $\mathcal{E}_i$ contains only non-superimposed edges. Further, the edges in $\mathcal{E}_i$ may be grouped into three categories: Previously-existing mc0-edges that have not been modified, newly-created mc0-edges that have not undergone facet-neighbor analysis, and class-0 conflict edges that were not superimposed.

Fixup newly-created mc0-edges.
For every $E_{\text{new}} \in \mathcal{E}_i$ whose mc0? field is set to `new`:

Add E_{new} to all contributing facets.
The previous step assured that the mc0-partial-facets field of E_{new} contains the set $\mathcal{F}$ of all of the facets that non-generically intersect to form E_{new}. Since E_{new} is newly-constructed, none of the facets in $\mathcal{F}$ have been completed, so all belong in E_{new}'s mc0-partial-facets field. To insure that later facet constructions access E_{new} for proper mc0-analysis, add E_{new} to the edges field of every facet in $\mathcal{F}$.

Set E_{new}'s facet-neighbors.
Using the geometry of the non-generic contact condition, identify the facets in $\mathcal{F}$ that lie on the obstacle surface, rather than buried in the interior. There are three possibilities:

Zero surface facets. E_{new} is a non-manifold spur edge, so call *isolate-edge*(E_{new}, F).

Two surface facets. E_{new} is a manifold edge, and the left and right adjacent facets F_{left} and F_{right} may be identified. Set E_{new}'s $\text{facet}_{\text{left}}$ field to F_{left}, and set E_{new}'s $\text{facet}_{\text{right}}$ field to F_{right}.

Four surface facets. E_{new} is a non-manifold 4-neighbor edge, so isolate the edge by calling the procedure *isolate-edge*(E_{new}, F).

Set the mc0? field of E_{new} to `old`.
endfor

endfor

Discontinue F's status as a partially-constructed facet.
At this point, all mc0-edges in the edges field of F have an mc0? field set to `old`. Further, F appears in the mc0-partial facets field of every mc0-edge it contains. After this procedure returns, the calling procedure will complete F by deleting unreachable features, *et cetera.* Thus F is no longer needed in the mc0-partial-facets field of any mc0-edge, and similarly, F no longer requires the presence of mc0-edges that do not list F as a left or right neighbor. We now clear these previous side-effects.
For every edge E_{mc0} in the edges field of F whose mc0? field is set to `old`:

Remove F from the mc0-partial-facets field of E_{mc0}.

If F is not in the $\text{facet}_{\text{left}}$ or $\text{facet}_{\text{right}}$ field of E_{mc0}, remove E_{mc0} from the edges field of F.

endfor

Return.

merge-collinear-class-0-edges ($\mathcal{E}_i$)

Identify previously-existing mc0-edges.
Partition $\mathcal{E}_i$ into $\mathcal{E}_{\text{mc0}}$ and $\mathcal{E}_{\text{conflict}}$, where $\mathcal{E}_{\text{mc0}}$ contains edges whose mc0? field is set to `old`, and $\mathcal{E}_{\text{conflict}}$ contains edges whose mc0? field is set to false. The edges in $\mathcal{E}_{\text{mc0}}$ are mc0-edges created during previous facet constructions, and should not be changed.

Merge conflict edges that overlap existing mc0-edges.
For every $E_{\text{mc0}} \in \mathcal{E}_{\text{mc0}}$:

Copy $\mathcal{E}_{\text{conflict}}$ to produce $\mathcal{E}'_{\text{conflict}}$.

For every $E_{\text{conflict}} \in \mathcal{E}'_{\text{conflict}}$:

If E_{mc0} and E_{conflict} share a finite overlap, then

Call the procedure *merge-superimposed-class-0-edges*$_{\text{mc0}}(E_{\text{mc0}}, E_{\text{conflict}}, \mathcal{E}_{\text{conflict}})$. This procedure merges E_{mc0} and E_{conflict}, preserving the c-edge E_{mc0}. Since E_{mc0} is a previously-existing mc0-edge, it is not modified by this procedure. However, E_{conflict} may be deleted, moved, or split during the merge operation; $\mathcal{E}_{\text{conflict}}$ and the features adjacent to E_{conflict} are updated to reflect these modifications. E_{mc0} and the remaining portions of E_{conflict} are topologically linked by this procedure, unless the isolated? field of E_{mc0} is true.

endif

endfor

endfor

Merge remaining superimposed edges to produce new mc0-edges.
At this point, $\mathcal{E}_{\text{conflict}}$ contains a set of collinear conflict edges that do not overlap any previously-existing mc0-edge. The algorithm now merges these edges wherever they overlap.
While there is a pair of edges (E_1, E_2) in $\mathcal{E}_{\text{conflict}}$ that share a finite overlap:

Call the procedure *merge-superimposed-class-0-edges*$_{\text{normal}}(E_1, E_2, \mathcal{E}_{\text{conflict}})$. This procedure merges E_1 and E_2, creating a new mc0-edge E_{new}, with its mc0? field set to `new`. The facet-neighbors of E_{new} are left blank, and E_{new}'s mc0-partial-facets field is set to contain all of the facets that have intersected E_{new} thus far. The procedure adds E_{new} to $\mathcal{E}_{\text{conflict}}$, which now contains both conflict edges and newly-created mc0-edges. E_1 and E_2 may have been deleted, moved, or split during the merge operation, and the procedure updates $\mathcal{E}_{\text{conflict}}$ and the adjacent features to reflect these changes. E_{new} is topologically linked to any remaining portions of E_1 and E_2.

endwhile

Update $\mathcal{E}_i$.
$\mathcal{E}_{\text{conflict}}$ now contains an ensemble of modified conflict edges and newly-created mc0-edges, none of which are superimposed. Return these edges by setting $\mathcal{E}_i$ to $\mathcal{E}_{\text{mc0}} \cup \mathcal{E}_{\text{conflict}}$. When this procedure returns, $\mathcal{E}_i$ will contain all of the old mc0-edges, newly-created mc0-edges, and conflict edges that result from merging the superimposed edges in $\mathcal{E}_i$.

Return.

merge-superimposed-class-0-edges$_{\text{mc0}}(E_{\text{mc0}}, E_{\text{conflict}}, \mathcal{E}_{\text{conflict}})$

Merge E_{mc0} and E_{conflict}, preserving E_{mc0}.
E_{mc0} is a previously-established mc0-edge, and so we know that it will not be split while merging with E_{conflict}. Therefore E_{mc0} must be contained in E_{conflict}, and we can treat E_{mc0} as the merged edge. Non-overlapping segments of E_{conflict} may remain; there are three possiblities:

Zero remaining segments.
E_{conflict} is subsumed by E_{mc0}, so remove E_{conflict} from $\mathcal{E}_{\text{conflict}}$ and *discard-edge*(E_{conflict}).

One remaining segment.
Set the endpoints of E_{conflict} to represent the segment of E_{conflict} that doesn't overlap E_{mc0}.

Two remaining segments.
E_{conflict} strictly contains E_{mc0}, and two segments of E_{conflict} remain on either side of E_{mc0}. Copy E_{conflict} to create E'_{conflict}, with the same fields as E_{conflict}. Set the endpoints of E_{conflict} and E'_{conflict} to represent the two segments of E_{conflict} that don't overlap E_{mc0}. Add E'_{conflict} to $\mathcal{E}_{\text{conflict}}$, and to the left and right facets neighboring E_{conflict}.

endif

If E_{mc0} is not isolated, topologically connect it to the remaining segments.
If the isolated? field of E_{mc0} is set to false, then

If E_{conflict} was not discarded, topologically connect E_{mc0} and E_{conflict}.

If E'_{conflict} exists, topologically connect E_{mc0} and E'_{conflict}.

endif

Return.

merge-superimposed-class-0-edges$_{\text{normal}}(E_1, E_2, \mathcal{E}_{\text{conflict}})$

E_1 and E_2 may be either conflict-edges or newly-formed mc0-edges. The superimposed portion of these edges will produce a newly-formed mc0-edge E_{new}, which will be added to $\mathcal{E}_{\text{conflict}}$. The facet-neighbors of E_{new} will be left blank, and the mc0-partial-facets field will contain all of the facets associated with E_1 and E_2. Non-overlapping portions of E_1 and E_2 may remain.

Form E_{new}.
Create a new c-edge E_{new} and set its fields as follows:

Set E_{new}'s endpoints and metric information fields to represent the overlapping segment of E_1 and E_2.

Set E_{new}'s mc0? field to `new`.

Set E_{new}'s contact-set to the union of the contact-sets for E_1 and E_2.

Set E_{new}'s mc0-partial-facets field to the union of the facets contained in the facet$_{\text{left}}$, facet$_{\text{right}}$, and mc0-partial-facets fields of E_1 and E_2.

Add E_{new} to $\mathcal{E}_{\text{conflict}}$.

Fixup E_1.
Part of E_1 overlaps E_{new}; non-overlapping segments of E_1 may remain. There are three possibilities:

Zero remaining segments.
E_1 is subsumed by E_{new}, so remove E_1 from $\mathcal{E}_{\text{conflict}}$ and *discard-edge*(E_1).

One remaining segments.
Set E_1's endpoints to represent the segment of E_1 that doesn't overlap E_{new}.

Two remaining segments.
Copy E_1 to create E_1'. Set the endpoints of E_1 and E_1' to represent the segments of E_1 that don't overlap E_{new}. Add E_1' to $\mathcal{E}_{\text{conflict}}$, and to the left and right facets neighboring E_1.

Fixup E_2.
Part of E_2 overlaps E_{new}; non-overlapping segments of E_2 may remain. There are three possibilities:

Zero remaining segments.
E_2 is subsumed by E_{new}, so remove E_2 from $\mathcal{E}_{\text{conflict}}$ and *discard-edge*(E_2).

One remaining segments.
Set E_2's endpoints to represent the segment of E_2 that doesn't overlap E_{new}.

Two remaining segments.
Copy E_2 to create E_2'. Set the endpoints of E_2 and E_2' to represent the segments of E_2 that don't overlap E_{new}. Add E_2' to $\mathcal{E}_{\text{conflict}}$, and to the left and right facets neighboring E_2.

Topologically connect E_{new}, E_1, E_1', E_2, and E_2'.
Connect E_{new} to the existing members of $\{E_1, E_1', E_2, E_2'\}$ to form a topological string of edges.

Return.

isolate-edge (E, F)

Set E's isolated? field to true.

If E's mc0? field is set to false, then

E is not a multiple-class-0 edge.
Since E is an ordinary local or conflict edge, no special side effects are required.

discard-edge (E)

Return.

else

E is a multiple-class-0 edge.
Since E is a multiple-class-0 edge, it must be retained in all of its contributing partial facets to assure that future mc0-analysis steps are performed properly. This implies that E should not be removed from facets other than F, and the endpoint vertices of E should be made independent to avoid accidental changes to E's metric properties.

Remove E from the edges field of F.

Let $V_{\min}$ and $V_{\max}$ be the vertices stored in E's $\text{vertex}_{\min}$ and $\text{vertex}_{\max}$ fields:

Remove E from the edges field of $V_{\min}$ and $V_{\max}$.
Copy $V_{\min}$ and $V_{\max}$ to create $V'_{\min}$ and $V'_{\max}$, with the same fields as $V_{\min}$ and $V_{\max}$.
Set the edges field of $V'_{\min}$ and $V'_{\max}$ to $\{E\}$.
Set E's $\text{vertex}_{\min}$ field to $V'_{\min}$, and set E's $\text{vertex}_{\max}$ field to $V'_{\max}$.

Return.

endif

discard-edge (E)

Remove E from the edges field of the facets stored in E's $\text{facet}_{\text{left}}$ and $\text{facet}_{\text{right}}$ fields.

Remove E from the edges field of the vertices stored in E's $\text{vertex}_{\min}$ and $\text{vertex}_{\max}$ fields.

Return.

c-obstacle

Polygons

moving-polygon fixed-polygon	The input polygons. Each polygon is represented by a data structure that includes slots for the feature parameters used in the *CO* algorithm. These parameters are calculated early and cached to avoid redundant computation.

Facet Array

facet-array	A three-dimensional array, indexed by a moving-polygon index i, a fixed-polygon index j, and a flag *ve-or-ev* $\in \{0, 1\}$. Each $(i, j,$ *ve-or-ev*$)$ array slot corresponds to a unique ve or ev contact condition. Each slot contains a set of facets that correspond to the slot's contact condition.

Control Information

incomplete-facet-set	The set of facets that are only partially constructed, and whose contact features are not globally convex. This set is implemented as a doubly-linked circular list, so that facets can be removed in constant time as they are completed.
incomplete-facet-set-constructed?	Boolean flag indicating whether the incomplete-facet-set has been filled in yet.

c-facet

Contact Information

contact-set	The set of feature-pairs in contact for this facet.
ve-or-ev moving-feature fixed-feature	Decomposed contact information. The moving-feature and fixed-feature fields point to features of the input polygons, and *ve-or-ev* $\in \{0, 1\}$ is a flag that indicates the contact type. This redundant information is included to avoid repeated parsing of the contact-set.

Metric Information

$f_x(p, \theta)$ $f_y(p, \theta)$	These functions return the (x, y) coordinates of points on the facet c-surface, given the parameters p and θ.
$f_p(x, y, \theta)$	This function returns the p-value of a point on the facet c-surface, given an (x, y, θ) configuration-space point. If the input point is not on the c-surface, then the function returns the p-value of the closest point on the c-surface in the specified θ-plane.
$\theta_{\min}$ $\theta_{\max}$	Bounding θ-values for the facet, used in quick θ-interval comparisons between facets.

Topological Information

edges	The set of obstacle edges bounding this facet. When the facet is only partially constructed, this is the set of candidate obstacle edges that have been posted to the facet.

Control Information

stage	A flag indicating the state of this facet in the computational process. One of `impossible`, `partial`, or `complete`.
conflict-possible?	Boolean flag indicating whether to consider non-local conflicts when completing this facet. This flag is set to true unless the contact edge and vertex are both globally convex.
incomplete-facet-set-entry	This points to the data object containing this facet in the incomplete-facet-set. When this facet is completed, it is removed from the set to avoid future conflict tests.
local-contacts-constructed	The set of local constraints for this facet that have been analyzed thus far in the computation, expressed as an ee or vv contact-pair. A local constraint entry will appear in this field even if no local edges due to the constraint were posted to this facet, or if a local edge was posted and then removed during the completion of the edge's opposite facet.
obstacle	The configuration-obstacle containing this facet.

c-edge

Contact Information

contact-set	The set of feature-pairs in contact for this edge.

Metric Information

class	One of 0, 1, 2, or 3.
$\theta_{\text{class-0}}$	If this is a class-0 edge, then this is the θ-value of the plane containing the edge.
coefficients_x coefficients_y	If this is a non-class-0 edge, then these are the coefficients of the edge function canonical form. Each of these fields is a list $(a\ b\ c\ d\ e\ f\ g\ h\ i\ j)$.
$f_x(p\text{-}or\text{-}\theta)$ $f_y(p\text{-}or\text{-}\theta)$	These functions return the (x, y) coordinates of points on the unbounded curve containing the edge. These are functions of p if this is a class-0 edge; otherwise, they are functions of θ.
$f'_x(\theta)$ $f'_y(\theta)$	If this is a non-class-0 edge, then these functions return the $\frac{dx}{d\theta}$ and $\frac{dy}{d\theta}$ derivatives of the unbounded curve containing the edge, given an input θ-value.
$f_p(x, y)$	If this is a class-0 edge, then this function returns the p-value of a point on the line containing the edge, given an (x, y) point. If the input point is not on the line, then the function returns the p-value of the closest point on the line.
$p_{\min}$ $p_{\max}$	If this is a class-0 edge, then these are the bounding p-values for the edge. For non-class-0 edges, the bounding θ-values are fetched from $\text{vertex}_{\min}$ and $\text{vertex}_{\max}$.

Topological Information

$\text{vertex}_{\min}$ $\text{vertex}_{\max}$	The edge endpoints. $\text{vertex}_{\min}$ is at the end of the edge that has the minimum edge parameter value (p or θ), while $\text{vertex}_{\max}$ is at the end with the maximum parameter value. Zero-length edges are never constructed.
$\text{facet}_{\text{left}}$ $\text{facet}_{\text{right}}$	The facets adjacent to this edge. The left and right directions are defined as follows: If you stand on the surface of the configuration-obstacle at $\text{vertex}_{\min}$ and look toward $\text{vertex}_{\max}$, then $\text{facet}_{\text{left}}$ is on your left.

Control Information

mc0?	One of `new`, `old`, or false. `new` indicates that this is a newly-constructed multiple-class-0 edge. `old` indicates that this is a previously-constructed multiple-class-0 edge, and false indicates that this is not a multiple-class-0 edge.
mc0-partial-facets	If this edge is a multiple-class-0 edge, then this is the set of facets that *(i)* contribute to the mc0-edge, and *(ii)* have not been completed. These facets should be updated if the edge is split by an intersection with a non-class-0 edge.
isolated?	Boolean flag indicating whether this edge has been isolated.
obstacle	The configuration-obstacle containing this edge.

c-vertex

Contact Information

contact-set	The set of feature-pairs in contact for this vertex. This field is lazy-evaluated, and cannot be constructed until all adjacent edges have been identified.

Metric Information

x	The (x, y, θ) coordinates of this vertex.
y	
θ	

Topological Information

edges	The set of edges adjacent to this vertex. Some edges may be missing from this set when the obstacle is only partially constructed. If all facets neighboring these edges have been completed, then no edges are missing.

Control Information

obstacle	The configuration-obstacle containing this vertex.

Appendix 6: Test Data

In this appendix, linear quantities are given in millimeters, and angular quantities are given in radians.

Tolerance Magnitudes and Units

Except for minor tuning parameters used to improve the comparison of numerically-sensitive configuration obstacle edge endpoints, the programs developed in this thesis uniformly apply two tolerances:

$$\begin{aligned} \epsilon_{xy} &= 0.05\,\text{mm} \\ \epsilon_{\theta} &= 0.0001\,\text{radians} \end{aligned}$$

where ϵ_{xy} is used for comparisons of linear quantities, and ϵ_{θ} is used for comparison of angular quantities. The numerical procedures described in Appendix 2 refine their roots to tolerances of 0.0000002 radians for the *CO* program, and 0.00001 radians for the *STATIC* program.

Polygons Used in the Physical Experiments

$[(31.6, 0.0)\ (0.0, 63.2)\ (23.7, 126.3)\ (-47.4, 126.3)\ (-47.4, -47.4)\ (49.7, -47.4)\ (49.7, 0.0)]$

$[(-17.5, -14.7)\ \ (4.6, -14.0)\ \ (-7.2, -11.6)\ \ (28.3, 33.4)]$

$[(-132.7, 126.8)\ \ (-112.7, -183.2)\ \ (-22.7, -283.2)\ \ (67.3, -183.2)\ \ (87.3, 126.8)$
$(567.3, 126.8)\ \ (367.3, 156.8)\ \ (-232.7, 156.8)\ \ (-432.7, 126.8)]$

} Fabricated in $1/4''$ delrin (a plastic similar to nylon).

$[(-25.8, -15.5)\ \ (-0.2, -15.5)\ \ (26.0, 31.1)]$ } Fabricated in $1/4''$ plexiglas.

$[(-96.4, -103.4)\ \ (-120.2, -139.3)\ \ (-105.0, -143.7)]$
$[(413.2, 266.9)\ \ (444.6, 227.9)\ \ (474.6, 229.0)]$

} Fabricated in $5/8''$ nylon, and affixed to a posterboard surface.

The first polygon is the pushing finger used in the example in Part I, and the first two experiments. The handles A and B are located at $(-9.4, 104.3)$ and $(-9.7, 29.9)$, respectively. Experiments #1, #3, and #4 assumed that the coefficient of friction was in the interval $[0.23, 0.48]$, while experiment #2 assumed the interval $[0.29, 0.48]$. These values were obtained by laboratory measurement of the kinetic and static friction coefficients over a series of trials.

The extremal support friction distributions used in experiment #1 were obtained by either attaching legs near the outer vertices of the second polygon, or gluing a small paper patch under the center of mass of the second polygon. These modifications concentrate the support pressure distribution either far from or close to the center of mass.

Polygon Appearing in Figure 84

This polygon causes the *CO* program to fail:

$[(-287.29, -59.95)\ \ (158.72, -23.54)\ \ (-320.21, -214.91)\ \ (-121.55, -430.06)$
$(-72.48, -203.53)\ \ (437.14, 17.26)\ \ (-39.38, 71.15)\ \ (-483.23, 151.90)]$

Test Polygons Appearing in Figure 14

1 : $[(0,0)\ (100,0)\ (50,69)]$
2 : $[(-100,-66.66666666)\ (0,-66.66666666)\ (100,133.33333333)]$
3 : $[(-150,0)\ (150,0)\ (0,69)]$
4 : $[(111.53603,-272.41943)\ (244.26998,272.41951)\ (-244.26994,90.05493)]$
5 : $[(-300,-200)\ (0,-200)\ (300,400)]$
6 : $[(-400,0)\ (-400,-100)\ (400,0)]$
7 : $[(-400,0)\ (400,0)\ (-400,100)]$
8 : $[(-219.61327,-138.25968)\ (219.61327,-138.25968)\ (219.61327,138.25968)\ (-219.61327,138.25968)]$
9 : $[(230,-240)\ (290,-150)\ (-230,240)\ (-290,150)]$
10 : $[(280,-200)\ (480,200)\ (-480,200)\ (-350,-200)]$
11 : $[(60,-15)\ (30,-230)\ (180,-10)\ (-180,230)]$
12 : $[(-390,210)\ (170,-90)\ (-500,-210)\ (500,-130)]$
13 : $[(-156,-55)\ (-35,-115)\ (35,-115)\ (156,-55)\ (0,115)]$
14 : $[(230,-200)\ (300,200)\ (180,-90)\ (-130,200)\ (-300,200)\ (-170,-200)]$
15 : $[(230,-200)\ (300,-15)\ (180,-90)\ (-130,200)\ (-300,200)\ (-170,-200)]$
16 : $[(230,-200)\ (300,200)\ (180,-90)\ (150,200)\ (-600,200)\ (-470,-200)]$
17 : $[(230,-200)\ (300,200)\ (180,-90)\ (150,200)\ (-800,200)\ (-670,-200)]$
18 : $[(100,0)\ (250,200)\ (-250,200)\ (-100,0)\ (-250,-200)\ (250,-200)]$
19 : $[(280,-200)\ (480,200)\ (380,200)\ (0,-90)\ (-300,200)\ (-480,200)\ (-350,-200)]$
20 : $[(280,-200)\ (480,200)\ (140,200)\ (70,-130)\ (-30,200)\ (-480,200)\ (-300,-200)]$
21 : $[(200,300)\ (450,300)\ (500,0)\ (550,350)\ (-650,350)\ (-650,0)\ (-100,0)]$
22 : $[(-100,0)\ (-650,0)\ (-650,-350)\ (550,-350)\ (500,0)\ (450,-300)\ (200,-300)]$
23 : $[(280,-200)\ (480,200)\ (380,200)\ (40,-120)\ (-40,-120)\ (-350,200)\ (-480,200)\ (-350,-200)]$
24 : $[(280,-200)\ (480,200)\ (280,200)\ (250,-140)\ (15,-140)\ (-15,200)\ (-480,200)\ (-340,-200)]$
25 : $[(280,-200)\ (480,200)\ (380,200)\ (80,-120)\ (-40,-120)\ (-350,200)\ (-480,200)\ (-350,-200)]$
26 : $[(280,-200)\ (480,200)\ (150,200)\ (130,-140)\ (0,-140)\ (-40,200)\ (-480,200)\ (-340,-200)]$
27 : $[(280,-200)\ (480,200)\ (380,200)\ (8,-120)\ (-8,-120)\ (-350,200)\ (-480,200)\ (-350,-200)]$
28 : $[(280,-200)\ (480,200)\ (260,200)\ (230,-140)\ (20,-140)\ (-10,200)\ (-480,200)\ (-340,-200)]$
29 : $[(280,-200)\ (480,200)\ (265,200)\ (265,-140)\ (0,-140)\ (0,200)\ (-480,200)\ (-340,-200)]$
30 : $[(300,-182.3)\ (500,-182.3)\ (500,317.7)\ (-500,317.7)\ (-500,-182.3)\ (-300,-182.3)\ (-300,-382.3)\ (300,-382.3)]$
31 : $[(-187,0)\ (-25,0)\ (0,-150)\ (250,-150)\ (225,0)\ (200,-125)\ (25,-125)\ (0,25)\ (-187,25)]$
32 : $[(500,-50)\ (500,450)\ (-50,450)\ (0,69)\ (50,400)\ (350,400)\ (100,0)\ (-500,0)\ (-500,-50)]$
33 : $[(500,-50)\ (500,450)\ (-50,450)\ (0,69)\ (50,400)\ (450,400)\ (450,0)\ (-500,0)\ (-500,-50)]$
34 : $[(0,0)\ (50,50)\ (150,50)\ (150,-50)\ (100,-100)\ (100,-200)\ (200,-200)\ (200,100)\ (-100,100)\ (-100,0)]$
35 : $[(250,-275)\ (250,175)\ (-10,175)\ (0,-50)\ (10,150)\ (225,150)\ (225,-250)\ (-225,-250)\ (-225,-50)$
$(-475,-50)\ (-475,-275)]$
36 : $[(-309.07,-131.99)\ (-165.63,-406.46)\ (-189.38,-281.64)\ (406.51,107.43)]$
37 : $[(-151.22,343.48)\ (197.79,398.91)\ (-240.58,489.28)\ (-301.94,-341.46)\ (-140.25,-315.07)\ (53.92,329.54)]$
38 : $[(110.85,-77.42)\ (117.97,362.37)\ (-236.04,172.44)\ (104.38,-177.82)]$
$[(-379.03,-454.36)\ (-318.37,-399.63)\ (-232.91,-253.30)]$
39 : $[(442.42,-251.31)\ (417.84,451.35)\ (-128.69,335.42)\ (134.96,176.32)\ (-260.89,-14.23)\ (-462.42,-286.21)$
$(430.07,-464.09)\ (-92.49,-267.67)]$
40 : $[(-287.29,-59.95)\ (158.72,-23.54)\ (-320.21,-214.91)\ (-121.55,-430.06)\ (-72.48,-203.53)\ (437.14,17.26)$
$(-39.38,71.15)\ (-483.23,73.49)]$

Preimage Points

The 25 sample (x_i, y_i, θ_i) points shown in Figure 16 are:

$(-175.0, 627.2, 5.662)$	$(-176.8, 626.3, 5.689)$	$(-174.4, 625.9, 5.689)$	$(-174.3, 626.0, 5.689)$	$(-174.6, 624.7, 5.715)$
$(-174.2, 625.2, 5.715)$	$(-174.7, 625.5, 5.715)$	$(-174.1, 624.2, 5.741)$	$(-174.7, 624.6, 5.741)$	$(-179.7, 624.3, 5.741)$
$(-174.6, 623.7, 5.767)$	$(-180.9, 623.2, 5.767)$	$(-174.7, 622.5, 5.767)$	$(-182.0, 622.0, 5.793)$	$(-174.6, 621.4, 5.793)$
$(-173.8, 622.5, 5.793)$	$(-175.7, 620.3, 5.820)$	$(-174.2, 621.8, 5.820)$	$(-174.4, 622.0, 5.820)$	$(-183.9, 619.5, 5.846)$
$(-177.2, 619.3, 5.846)$	$(-175.8, 620.8, 5.846)$	$(-182.9, 618.6, 5.872)$	$(-177.4, 619.6, 5.872)$	$(-179.7, 618.2, 5.898)$

These 25 points combined with 2 support pressure distributions, 3 object positions (sampled at 120° intervals on the boundary of the position error ball), 2 object orientations, 2 robot orientations, and 2 robot pushing directions to give $25 \cdot 2 \cdot 3 \cdot 2 \cdot 2 \cdot 2 = 1200$ pushing trials with extremal uncertainty conditions.

Bibliography

[Abel *et al.* 1985] J. M. Abel, W. Holzmann, and J. M. McCarthy. On grasping planar objects with two articulated fingers. *IEEE Journal of Robotics and Automation*, RA-1(4):211–214, December 1985.

[Aboaf *et al.* 1989] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. Task-level robot learning: Juggling a tennis ball more accurately. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1290–1295, May 1989.

[Aho *et al.* 1983] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Massachusetts, 1983.

[Allen 1988] P. K. Allen. Integrating vision and touch for object recognition tasks. *International Journal of Robotics Research*, 7(6):15–33, 1988.

[Asada and By 1985] H. Asada and A. B. By. Kinematic analysis of workpart fixturing for flexible assembly with automatically reconfigurable fixtures. *IEEE Journal of Robotics and Automation*, RA-1(2):86–94, June 1985.

[Avnaim and Boissonnat 1989] F. Avnaim and J. D. Boissonnat. Polygon placement under translation and rotation. *Informatique Théorique et Applications*, 23(1):5–28, 1989.

[Avnaim *et al.* 1988a] F. Avnaim, J. D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1656–1661, April 1988.

[Avnaim *et al.* 1988b] F. Avnaim, J. D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. Rapports de Recherche No. 890, INRIA - Sophia Antipolis, August 1988.

[Ayache and Faugeras 1988] N. Ayache and O. D. Faugeras. Building, registrating, and fusing noisy data maps. *International Journal of Robotics Research*, 7(6):45–65, December 1988.

[Bajaj and Kim 1988] C. Bajaj and M. S. Kim. Generation of configuration-space obstacles: The case of a moving sphere. *IEEE Journal of Robotics and Automation*, 4(1):94–99, February 1988.

[Baker *et al.* 1985] B. S. Baker, S. Fortune, and S. Grosse. Stable prehension with a multi-fingered hand. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 570–575, March 1985.

[Balorda 1990] Z. Balorda. Reducing uncertainty of objects by robot pushing. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1051–1056, May 1990.

[Barber *et al.* 1986] J. Barber, R. A. Volz, R. Desai, R. Rubinfeld, B. Schipper, and J. Wolter. Automatic two-fingered grip selection. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 890–896, April 1986.

[Barraquand *et al.* 1990] J. Barraquand, B. Langlois, and J. Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In H. Miura and S. Arimoto, editors, *Robotics Research: The Fifth International Symposium*, pages 435–444, Cambridge, Massachusetts, 1990. MIT Press.

[Baumgart 1974] B. G. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University Department of Computer Science, October 1974.

[Bolles and Paul 1973] R. Bolles and R. Paul. The use of sensory feedback in a programmable assembly system. Technical Report AIM-220, Stanford Artificial Intelligence Laboratory, 1973.

[Boothroyd *et al.* 1972] G. Boothroyd, A. H. Redford, C. Poli, and L. E. Murch. Statistical distributions of natural resting aspects of parts for automatic handling. *Manufacturing Engineering Transactions, Society of Manufacturing Automation*, 1:93–105, 1972.

[Brach 1984] R. M. Brach. Friction, restitution, and energy loss in planar collisions. *Journal of Applied Mechanics*, 51:164–170, 1984.

[Briggs 1989] A. J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. Technical Report TR 89-980, Cornell University Department of Computer Science, March 1989.

[Brockett 1984] R. W. Brockett. Smooth multimode control systems. In L. R. Hunt and C. F. Martin, editors, *Proceedings of the Berkeley-Ames Conference on Nonlinear Problems in Control and Fluid Dynamics*, pages 103–110, Brookline, Massachusetts, 1984. Math Sci Press.

[Brooks and Lozano-Pérez 1985] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15:224–233, March/April 1985.

[Brooks 1982] R. A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(4):29–68, Winter 1982.

[Brooks 1983] R. A. Brooks. Planning collision-free motions for pick and place operations. *International Journal of Robotics Research*, 2(4):19–44, Winter 1983.

[Brost and Mason 1990] R. C. Brost and M. T. Mason. Graphical analysis of planar rigid-body dynamics with multiple frictional contacts. In H. Miura and S. Arimoto, editors, *Robotics Research: The Fifth International Symposium*, pages 293–300, Cambridge, Massachusetts, 1990. MIT Press.

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

[Brost 1989] R. C. Brost. Computing metric and topological properties of configuration-space obstacles. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 170–176, May 1989.

[Buckley 1989] S. J. Buckley. Teaching compliant motion strategies. *IEEE Transactions on Robotics and Automation*, 5(1):112–118, February 1989.

[Buhler *et al.* 1990] M. Buhler, D. E. Koditschek, and P. J. Kindlmann. Planning and control of robotic juggling tasks. In H. Miura and S. Arimoto, editors, *Robotics Research: The Fifth International Symposium*, pages 321–332, Cambridge, Massachusetts, 1990. MIT Press.

[Caine *et al.* 1989] M. E. Caine, T. Lozano-Pérez, and W. P. Seering. Assembly strategies for chamferless parts. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 472–477, May 1989.

[Canny and Reif 1987] J. F. Canny and J. Reif. New lower bound techniques for robot motion planning. In *Proceedings, 28th Annual Symposium on Foundations of Computer Science*, pages 49–60. IEEE, October 1987.

[Canny 1986] J. F. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):200–209, March 1986.

[Canny 1987] J. F. Canny. A new algebraic method for robot motion planning and real geometry. In *Proceedings, 28th Annual Symposium on Foundations of Computer Science*, pages 39–48. IEEE, October 1987.

[Canny 1989] J. F. Canny. On computability of fine motion plans. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 177–182, May 1989.

[Chelpanov and Kolpashnikov 1983] I. B. Chelpanov and S. N. Kolpashnikov. Problems with the mechanics of industrial robot grippers. *Mechanism and Machine Theory*, 18(4):295–299, 1983.

[Christiansen *et al.* 1990] A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning reliable manipulation strategies without initial physical models. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1224–1230, May 1990.

[Coulomb 1781] C. A. Coulomb. Théorie des machines simples en ayant égard au frottement de leurs parties et à la roider des cordages. In *Mémoires de Mathématique et de Physique présentés à l'Academie Royale des Sciences*, Paris, 1781.

[Cutkosky 1985] M. R. Cutkosky. *Grasping and Fine Manipulation for Automated Manufacturing.* PhD thesis, Carnegie Mellon University Department of Mechanical Engineering, January 1985.

[Desai 1989] R. S. Desai. *On Fine Motion in Mechanical Assembly in Presence of Uncertainty.* PhD thesis, University of Michigan Department of Mechanical Engineering, 1989.

[Donald and Pai 1990] B. R. Donald and D. K. Pai. On the motion of compliantly-connected rigid bodies in contact, part II: A system for analyzing designs for assembly. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1756–1762, May 1990.

[Donald 1987a] B. R. Donald. *Error Detection and Recovery in Robotics.* Springer-Verlag, New York, 1987.

[Donald 1987b] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31:295–353, 1987.

[Donald 1990] B. R. Donald. Planning multi-step error-detection and recovery strategies. *International Journal of Robotics Research*, 9(1):3–60, February 1990.

[Dooley and McCarthy 1990] J. R. Dooley and J. M. McCarthy. Parameterized descriptions of the joint space obstacles for a 5R closed chain robot. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1536–1541, May 1990.

[Drake 1977] S. H. Drake. *Using Compliance in Lieu of Sensory Feedback for Automatic Assembly.* PhD thesis, MIT Department of Mechanical Engineering, September 1977.

[Dufay and Latombe 1984] B. Dufay and J. Latombe. An approach to automatic robot programming based on inductive learning. *International Journal of Robotics Research*, 3(4):3–20, Winter 1984.

[Dunn and Segen 1988] G. B. Dunn and J. Segen. Automatic discovery of robotic grasp configurations. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 396–401, April 1988.

[Durrant-Whyte 1988] H. F. Durrant-Whyte. Sensor models and multisensor integration. *International Journal of Robotics Research*, 7(6):97–113, December 1988.

[Elfes 1987] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, June 1987.

[Englert and Wright 1986] P. J. Englert and P. K. Wright. Applications of artificial intelligence and the design of fixtures for automated manufacturing. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 345–351, April 1986.

[Erdmann and Mason 1986] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. In *Proceedings, IEEE International Conference on Robotics and Automation*, 1986.

[Erdmann and Mason 1988] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, August 1988.

[Erdmann 1984] M. A. Erdmann. On motion planning with uncertainty. Master's thesis, MIT Department of Electrical Engineering and Computer Science, August 1984.

[Erdmann 1986] M. A. Erdmann. Using backprojections for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):19–45, Spring 1986.

[Erdmann 1989] M. A. Erdmann. *On Probabilistic Robot Strategies.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, August 1989.

[Ernst 1961] H. A. Ernst. *MH-1, A Computer-Operated Mechanical Hand.* PhD thesis, MIT Department of Electrical Engineering, December 1961.

[Fahlman 1974] S. E. Fahlman. A planning system for robot construction tasks. *Artificial Intelligence*, 5:1–49, 1974.

[Faltings 1987] B. Faltings. *Qualitative Place Vocabularies for Mechanisms in Configuration Space*. PhD thesis, University of Illinois Department of Computer Science, July 1987.

[Fearing 1986] R. S. Fearing. Simplified grasping and manipulation with dextrous robot hands. *IEEE Journal of Robotics and Automation*, RA-2(4):188–195, December 1986.

[Featherstone 1986] R. Featherstone. The dynamics of rigid body systems with multiple concurrent contacts. In O. Faugeras and G. Giralt, editors, *Robotics Research: The Third International Symposium*, pages 189–196, Cambridge, Massachusetts, 1986. MIT Press.

[Fikes and Nilsson 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fitzgerald *et al.* 1981] W. Fitzgerald, F. Gracer, and R. Wolfe. GRIN: Interactive graphics for modeling solids. *IBM Journal of Research and Development*, 25(4):281–294, 1981.

[Forbus 1981] K. D. Forbus. A study of qualitative and geometric knowledge in reasoning about motion. Technical Report 615, MIT Artificial Intelligence Laboratory, 1981.

[Forbus 1988] K. D. Forbus. Qualitative physics: Past, present, and future. In H. Shrobe, editor, *Exploring Artificial Intelligence*, pages 239–296. Morgan Kaufmann, San Mateo, California, 1988.

[Ge and McCarthy 1989] Q. Ge and J. M. McCarthy. Equations for boundaries of joint obstacles for planar robots. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 164–169, May 1989.

[Goertz 1963] R. C. Goertz. Manipulators used for handling radioactive materials. In E. Bennett, J Degan, and J. Spiegel, editors, *Human Factors in Technology*, pages 425–443. McGraw-Hill, New York, 1963.

[Goldberg and Mason 1990] K. Y. Goldberg and M. T. Mason. Bayesian grasping. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1264–1269, May 1990.

[Goldberg 1990] K. Y. Goldberg. *Stochastic Plans for Robotic Manipulation*. PhD thesis, Carnegie Mellon University School of Computer Science, November 1990.

[Goto *et al.* 1980] T. Goto, K. Takeyasu, and T. Inoyama. Control algorithm for precision insert operation robots. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(1):19–25, 1980.

[Goyal 1989] S. Goyal. *Planar Sliding of a Rigid Body with Dry Friction: Limit Surfaces and Dynamics of Motion*. PhD thesis, Cornell University Department of Mechanical Engineering, January 1989.

[Gross forthcoming] K. P. Gross. *Concept Acquisition Through Feature Evolution and Experimentation*. PhD thesis, Carnegie Mellon University School of Computer Science, forthcoming.

[Grossman and Blasgen 1975] D. D. Grossman and M. W. Blasgen. Orienting mechanical parts by computer-controlled manipulator. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5(5):561–565, September 1975.

[Hanafusa and Asada 1977] H. Hanafusa and H. Asada. Stable prehension by a robot hand with elastic fingers. In *Proceedings, 7th International Symposium of Industrial Robots*, pages 361–368, October 1977. Reprinted in M. Brady, *et al.,* editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.

[Hilbert and Cohn-Vossen 1932] D. Hilbert and S. Cohn-Vossen. *Anschauliche Geometrie*. Göttingen, 1932. Republished as *Geometry and the Imagination,* New York: Chelsea Publishing Company, 1952.

[Hoffmann and Hopcroft 1986] C. M. Hoffmann and J. E. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3(3):194–206, June 1986.

[Hoffmann 1989] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, California, 1989.

[Hogan 1985] N. Hogan. Impedance control: An approach to manipulation. *Journal of Dynamic Systems, Measurement, and Control*, 107:1–24, March 1985.

[Ikeuchi *et al.* 1986] K. Ikeuchi, H. K. Nishihara, B. K. P. Horn, P. Sobalvarro, and S. Nagata. Determining grasp configurations using photomoteric stereo and the prism binocular stereo system. *International Journal of Robotics Research*, 5(1):46–65, Spring 1986.

[Inoue 1974] H. Inoue. Force feedback in precise assembly tasks. Memo 308, MIT Artificial Intelligence Laboratory, August 1974.

[Jameson and Leifer 1987] J. W. Jameson and L. J. Leifer. Automatic grasping: An optimization approach. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(5):806–814, September/October 1987.

[Jellet 1872] J. H. Jellet. *A Treatise on the Theory of Friction.* MacMillan, London, 1872.

[Jones and Lozano-Pérez 1990] J. L. Jones and T. Lozano-Pérez. Planning two-fingered grasps for pick-and-place operations on polyhedra. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 683–688, May 1990.

[Joskowicz and Addanki 1988] L. Joskowicz and S. Addanki. From kinematics to shape: An approach to innovative design. In *Proceedings, AAAI: The Seventh National Conference on Artificial Intelligence*, pages 347–352, August 1988.

[Joskowicz and Sacks 1990] L. Joskowicz and E. Sacks. Unifying kinematics and dynamics for the automatic analysis of machines. Technical Report RC 15573, IBM, March 1990.

[Kerr and Roth 1986] J. Kerr and B. Roth. Analysis of multifingered hands. *International Journal of Robotics Research*, 4(4):3–17, Winter 1986.

[Khatib 1986] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, Winter 1986.

[Latombe 1989] J. Latombe. Motion planning with uncertainty: On the preimage backchaining approach. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review I*, pages 53–70. MIT Press, Cambridge, Massachusetts, 1989.

[Li and Sastry 1988] Z. Li and S. S. Sastry. Task-oriented optimal grasping by multifingered robot hands. *IEEE Journal of Robotics and Automation*, 4(1):32–43, 1988.

[Lötstedt 1981] P. Lötstedt. Coulomb friction in two-dimensional rigid-body systems. *Zeitschrift für Angewandte Mathematik un Mechanik*, 61:605–615, 1981.

[Lozano-Pérez and Wesley 1979] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[Lozano-Pérez *et al.* 1984] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, Spring 1984.

[Lozano-Pérez *et al.* 1988] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournassoud, and A. Lanusse. Handey: A task-level robot system. In R. Bolles and B. Roth, editors, *Robotics Research: The Fourth International Symposium*, pages 29–36, Cambridge, Massachusetts, 1988. MIT Press.

[Lozano-Pérez 1976] T. Lozano-Pérez. The design of a mechanical assembly system. Memo 397, MIT Artificial Intelligence Laboratory, December 1976.

[Lozano-Pérez 1981] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(10):681–698, October 1981. Reprinted in M. Brady, *et al.,* editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.

[Lozano-Pérez 1987] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, June 1987.

[Lumelsky and Stepanov 1987] V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[MacMillan 1936] W. D. MacMillan. *Dynamics of Rigid Bodies.* McGraw-Hill, New York, 1936.

[Mani and Wilson 1985] M. Mani and R. D. W. Wilson. A programmable orienting system for flat parts. In *North American Manufacturing Research Institute Conference XIII*, 1985.

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

[Mason and Brost 1986] M. T. Mason and R. C. Brost. Automatic grasp planning: An operation space approach. In *Proceedings, 6th Symposium on Theory and Practice of Robots and Manipulators*, pages 321–328, September 1986.

[Mason and Wang 1988] M. T. Mason and Y. Wang. On the inconsistency of rigid-body frictional planar mechanics. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 524–528, April 1988.

[Mason 1981] M. T. Mason. Compliance and force control for computer coontrolled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(6):418–432, June 1981. Reprinted in M. Brady, *et al.,* editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.

[Mason 1982] M. T. Mason. *Manipulator Grasping and Pushing Operations.* PhD thesis, Massachusetts Institute of Technology, June 1982. Reprinted in *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, Massachusetts, 1985.

[Mason 1984] M. T. Mason. Automatic planning of fine motions: Correctness and completeness. In *Proceedings, IEEE International Conference on Robotics*, pages 492–503, March 1984.

[Mason 1985] M. T. Mason. The mechanics of manipulation. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 544–548, March 1985.

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

[Mason 1986b] M. T. Mason. On the scope of quasi-static pushing. In O. Faugeras and G. Giralt, editors, *Robotics Research: The Third International Symposium*, pages 229–233, Cambridge, Massachusetts, 1986. MIT Press.

[Matthies and Shafer 1987] L. H. Matthies and S. A. Shafer. Error modelling in stereo navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):239–248, 1987.

[Milenkovic 1988] V. J. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Carnegie Mellon University Department of Computer Science, July 1988.

[Mishra *et al.* 1987] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4):641–558, 1987.

[Montana 1988] D. J. Montana. The kinematics of contact and grasp. *International Journal of Robotics Research*, 7(3):17–32, June 1988.

[Moseley 1835] H. Moseley. On the equilibrium of the arch. *Transactions of the Cambridge Philosophical Society*, V:293–314, 1835.

[Natarajan 1988] B. K. Natarajan. The complexity of fine-motion planning. *International Journal of Robotics Research*, 7(2):36–42, April 1988.

[Natarajan 1989] B. K. Natarajan. Some paradigms for the automated design of parts feeders. *International Journal of Robotics Research*, 8(6):98–109, December 1989.

[Newell and Simon 1976] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, March 1976.

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

[Ohwovoriole *et al.* 1980] M. S. Ohwovoriole, J. W. Hill, and B. Roth. On the theory of single and multiple insertions in industrial assemblies. In *Proceedings, 10th International Symposium of Industrial Robots*, pages 545–558, March 1980.

[Ohwovoriole 1980] M. S. Ohwovoriole. An extension of screw theory and its application to the automation of industrial assemblies. Technical Report AIM-338, Stanford Artificial Intelligence Laboratory, April 1980.

[Park and Starr 1990] Y. C. Park and G. P. Starr. Grasp synthesis of polygonal objects. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1574–1580, May 1990.

[Pertin-Troccaz 1989] J. Pertin-Troccaz. Grasping: A state of the art. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review I*, pages 71–98. MIT Press, Cambridge, Massachusetts, 1989.

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

[Peshkin and Sanderson 1988b] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5):524–531, October 1988.

[Peshkin 1986] M. A. Peshkin. *Planning Robotic Manipulation Strategies for Sliding Objects*. PhD thesis, Carnegie Mellon University Department of Physics, November 1986.

[Pham *et al.* 1990] D. T. Pham, K. C. Cheung, and S. H. Yeo. Initial motion of a rectangular object being pushed or pulled. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1046–1050, May 1990.

[Pingle *et al.* 1974] K. Pingle, R. Paul, and R. Bolles. *Programmable Assembly: Three Short Examples*. Stanford Artificial Intelligence Laboratory, 1974. Film.

[Prescott 1923] J. Prescott. *Mechanics of Particles and Rigid Bodies*. Longmans, Greene, and Co., London, 1923.

[Raibert and Craig 1981] M. H. Raibert and J. J. Craig. Hybrid force/position control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102:126–133, June 1981. Reprinted in M. Brady, *et al.*, editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.

[Rajan *et al.* 1987] V. T. Rajan, R. Burridge, and J. T. Schwartz. Dynamics of a rigid body in frictional contact with rigid walls. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 671–677, March 1987.

[Requicha 1980] A. A. G. Requicha. Representations for rigid solids: Theory, models, and systems. *ACM Computing Surveys*, 12:437–464, 1980.

[Reuleaux 1876] F. Reuleaux. *The Kinematics of Machinery*. Macmillan, 1876. Republished by Dover, New York, 1963.

[Rimon and Koditschek 1989] E. Rimon and D. E. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 21–26, May 1989.

[Salisbury and Craig 1982] J. K. Salisbury and J. J. Craig. Articulated hands: Force control and kinematic issues. *International Journal of Robotics Research*, 1(1):4–17, Spring 1982.

[Salisbury 1982] J. K. Salisbury. *Kinematic and Force Analysis of Articulated Hands*. PhD thesis, Stanford University Department of Mechanical Engineering, May 1982. Reprinted in *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, Massachusetts, 1985.

[Sanderson 1984] A. C. Sanderson. Parts entropy methods for robotic assembly system design. In *Proceedings, IEEE International Conference on Robotics*, pages 600–608, March 1984.

[Schwartz and Sharir 1983] J. Schwartz and M. Sharir. On the piano movers problem I: The case of a two-dimensional rigid body moving amidst polygonal barriers. *Communications of Pure and Applied Mathematics*, 36:345–398, 1983.

[Simunovic 1975] S. Simunovic. Force information in assembly processes. In *Proceedings, 5th International Symposium of Industrial Robots*, pages 415–431, September 1975.

[Smith and Cheeseman 1986] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[Stansfield 1988] S. A. Stansfield. A robotic perceptual system using passive vision and active touch. *International Journal of Robotics Research*, 7(6):138–161, December 1988.

[Stefik 1981] M. J. Stefik. Planning with constraints (MOLGEN: Part I). *Artificial Intelligence*, 16:111–140, 1981.

[Stentz 1990] A. Stentz. Multi-resolution constraint modeling for mobile robot planning. In C. E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab*, pages 231–257. Kluwer Academic Publishers, Norwell, Massachusetts, 1990.

[Strip and Maciejewski 1990] D. R. Strip and A. A. Maciejewski. Archimedes: An experiment in automating mechanical assembly. In *International Symposium on Robotics and Manufacturing*, July 1990.

[Strip 1989] D. R. Strip. A passive mechanism for insertion of convex pegs. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 242–248, May 1989.

[Sturges 1988] R. H. Sturges, Jr. A three-dimensional assembly task quantification with application to machine dexterity. *International Journal of Robotics Research*, 7(4):34–78, August 1988.

[Sugihara and Iri 1989] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380–393, 1989.

[Sussman 1973] G. J. Sussman. A computational model of skill acquisition. Technical Report AI-TR-297, MIT Artificial Intelligence Laboratory, August 1973.

[Taylor and Rajan 1988] R. H. Taylor and V. T. Rajan. The efficient computation of uncertainty spaces for sensor-based robot planning. In *Proceedings of the IEEE Workshop on Intelligent Robots and Systems*, November 1988.

[Taylor *et al.* 1988] R. H. Taylor, M. T. Mason, and K. Y. Goldberg. Sensor-based manipulation planning as a game with nature. In R. Bolles and B. Roth, editors, *Robotics Research: The Fourth International Symposium*, pages 421–429, Cambridge, Massachusetts, 1988. MIT Press.

[Taylor 1976] R. H. Taylor. *A Synthesis of Manipulator Control Programs from Task-Level Specification*. PhD thesis, Stanford University Department of Computer Science, July 1976.

[Tournassoud and Jehl 1988] P. Tournassoud and O. Jehl. Motion planning for a mobile robot with a kinematic constraint. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1785–1790, April 1988.

[Trinkle *et al.* 1988] J. C. Trinkle, J. M. Abel, and R. P. Paul. An investigation of frictionless enveloping grasping in the plane. *International Journal of Robotics Research*, 7(3):33–51, June 1988.

[Udupa 1977] S. M. Udupa. Collision detection and avoidance in computer controlled manipulators. In *Proceedings, 5th International Joint Conference on Artificial Intelligence*, 1977.

[Volpe and Khosla 1990] R. Volpe and P. Khosla. A strategy for obstacle avoidance and approach using superquadric potential functions. In H. Miura and S. Arimoto, editors, *Robotics Research: The Fifth International Symposium*, pages 445–452, Cambridge, Massachusetts, 1990. MIT Press.

[Wang and Mason 1987] Y. Wang and M. Mason. Modeling impact dynamics for robotic operations. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 678–685, April 1987.

[Wang 1989] Y. Wang. Dynamics and planning of collisions in robotic manipulation. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 478–483, May 1989.

[Wesley *et al.* 1980] M. A. Wesley, T. Lozano-Pérez, L. I. Lieberman, M. A. Lavin, and D. D. Grossman. A geometric modeling system for automated mechanical assembly. *IBM Journal of Research and Development*, 24:64–74, 1980.

[Wesley 1980] M. A. Wesley. Construction and use of geometric models. In J. Encarnacao, editor, *Computer Aided Design: Lecture Notes in Computer Science No. 89*. Springer-Verlag, New York, 1980.

[Whitney 1977] D. E. Whitney. Force feedback control of manipulator fine motions. *Journal of Dynamic Systems, Measurement, and Control*, pages 91–97, June 1977.

[Whitney 1982] D. E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *Journal of Dynamic Systems, Measurement, and Control*, 104:65–77, March 1982. Reprinted in M. Brady, *et al.,* editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.

[Whitney 1987] D. E. Whitney. Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, 6(1):3–14, Spring 1987.

[Wolter *et al.* 1985] J. Wolter, R. A. Volz, and A. C. Woo. Automatic generation of gripping positions. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(2):204–213, March/April 1985.

[Yap 1987] C. K. Yap. Algorithmic motion planning. In J. T. Schwartz and C. K. Yap, editors, *Advances in Robotics, Volume I: Algorithmic and Geometric Aspects*, pages 95–143. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

[Young 1978] K. D. Young. Controller design for a manipulator using theory of variable structure systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(2):101–109, February 1978. Reprinted in M. Brady, *et al.,* editors, *Robot Motion: Planning and Control,* MIT Press, Cambridge, Massachusetts, 1982.