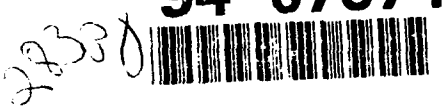
(2)

Technical Report 1464

Parallel Methods for Synthesizing Whole-Hand Grasps from Generalized Prototypes

Nancy S. Pollard

MIT Artificial Intelligence Laboratory



REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1994	technical report

4. TITLE AND SUBTITLE

Parallel Methods for Synthesizing Whole-Hand Grasps from Generalized Prototypes

5. FUNDING NUMBERS

N00014-91-J-4038
N00014-92-J-1814

6. AUTHOR(S)

Nancy S. Pollard

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Massachusetts Institute of Technology
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

AI-TR 1464

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Information Systems
Arlington, Virginia 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report addresses the problem of acquiring objects using articulated robotic hands. Standard grasps are used to make the problem tractable, and a technique is developed for generalizing these standard grasps to increase their flexibility to variations in the problem geometry. A generalized grasp description is applied to a new problem situation using a parallel search through hand configuration space, and the result of this operation is a global overview of the space of good solutions. The techniques presented in this report have been implemented, and the results are verified using the Salisbury three-finger robotic hand.

14. SUBJECT TERMS

robotics articulated hands
grasping whole-hand grasps
parallel grasp generalization

15. NUMBER OF PAGES
289

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

Parallel Methods for Synthesizing Whole-Hand Grasps from Generalized Prototypes

by

Nancy S. Pollard

Revised version of a thesis submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Massachusetts Institutee of Technology on January 7, 1994.

Abstract. Robotic hands are very flexible mechanisms. Because of this flexibility, it has been difficult to automate the process of acquiring objects using these robotic hands. Each new grasping problem is too complex to analyze without the use of good heuristics. Constraints due to target object geometry, environment geometry, hand kinematics and geometry, and a task description must all be considered when forming a solution.

One tempting approach to this problem is to contain the flexibility of the robot hand by developing standard routines for grasping common objects. Standard grasping routines greatly reduce the complexity of the grasp synthesis problem, but these routines tend to go too far, becoming inflexible to variation in the target object and environment geometries. If the range of application of any given grasping routine is small, a reasonably sized library of routines will not cover an acceptable space of problem situations.

This report describes one way in which standard grasping routines can be made more effective. A technique is developed for generalizing standard grasps and applying a generalized grasp description to new problem situations. The grasp generalization technique expands the range of application of a standard grasp to a wide variety of target object and environment geometries, while ensuring that the resulting grasps are appropriate for the intended task. The ability to generalize a standard grasp in this way greatly reduces the number of standard grasping routines that must be stored in any comprehensive grasp library.

A generalized grasp description is applied to a new problem situation using a parallel search through hand configuration space. The generalized grasp description allows the placement of each contact of a grasp to be independently optimized. This means that the process of assembling good hand configurations can be transformed into a dynamic programming algorithm. The many-dimensional space of hand configurations (fifteen dimensions in the examples of the report) is searched for optimal solutions by performing a small number of steps in a six-dimensional workspace.

The result of applying a generalized grasp description to a new problem situation is a space of wrist configurations from which high quality, collision-free grasps can be achieved. It is useful to have such a global solution, because this solution can be used to estimate the robustness of a grasp to errors in wrist placement, to measure the effect of obstacles on the size and shape of the space of good grasps, and to indicate whether any good grasps are possible at all. A global solution also provides a flexible starting point for a post-processing step that adds the constraint of robot arm kinematics to the problem and finds an approach path into one of the high quality, collision-free grasps that have been identified.

The techniques presented in this report have been implemented. Many simulated examples are shown throughout the report, and the results are verified using the Salisbury three-finger robotic hand.

Acknowledgments

I would like to thank my advisor, Tomás Lozano-Pérez, for his constant help and guidance, and for reading countless drafts, sometimes painfully unfinished, of my papers and thesis documents.

I would also like to thank the other members of my thesis committee, Eric Grimson and Ken Salisbury, for their many helpful comments and careful reading of this report. Both Eric and Ken have been sources of support and encouragement.

The hardware used in this report has required plenty of supervision and support. Thanks to Patrick O'Donnell for answering all my CM questions and for making sure the machine stayed up. Thanks to Ken Salisbury for the robotic hand, and to Dave Brock, Jose Robles, Sundar Narasimhan, and everyone else who kept the puma lab going.

David Siegel has been supportive in many ways, helping me work through the process of selecting a thesis topic, serving as a springboard for ideas, and commenting on multiple drafts of this report. He has also been a source of stability outside the lab. Thanks David, for making me believe I could finish!

My officemate, Maja Mataric, has been a day to day source of encouragement, sharing in my excitement in the good days, and helping lift me out of the bad days during the writing of this report. She is largely responsible for getting me to run again, and besides making the office a fun place to be, also made sure I got out of the office once in a while. Many thanks, Maja!

Many other friends have brightened these last few years at MIT, especially Mike Bolotski, Jean-Pierre Shott, Ira Haimowitz, Karen Sarachik and Joel Epstein, Sundar Narasimhan and Marie Lamb, Jose Robles, and Colin Angle. Thank you all!

And finally, thanks to my family for helping me to keep my perspective.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contact N00014-91-J-4038 and in part by the University Research Initiative under Office of Naval Research contract N00014-92-J-1814.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Our hands are very important tools. We use them to acquire and manipulate objects of a wide variety of shapes, sizes, and materials. We have robust strategies for manipulating oddly-shaped objects such as toy cows and airplanes, performing tricky tasks such as lifting coins from tables, and working with flexible objects such as clothes or shoelaces.

Robotic hands are very flexible mechanisms. It has been very difficult, however, to successfully automate the processes of acquiring and manipulating objects using today's robotic hands. One tempting solution is to develop standard routines for grasping common objects. Standard grasping routines solve part of the problem, and they may require very little computation, but they do not seem sufficiently robust to work in complex situations. At the opposite extreme, a thorough analysis of each grasping or manipulation task would allow more robust grasping strategies to be designed, but this type of analysis is too difficult to achieve in any reasonable amount of computation time. This report presents a hybrid solution to the problem of grasp synthesis, showing how some analysis of a specific grasping task can be used to determine how a standard grasp can be effectively applied in a non-standard situation.

1.1 Grasp Synthesis

This report considers the problem of identifying good grasps of a target object using geometric information. This problem can be stated as follows. Given:

- a geometric model of a target object,
- a geometric model of the environment,
- a task, and
- geometric and kinematic models of the robot,

find a grasp of the target object that is suitable for the given task.

In the problem illustrated in Figure 1.1, for example, the information available would include a geometric model of the hammer, as well as models of the pins and the wall

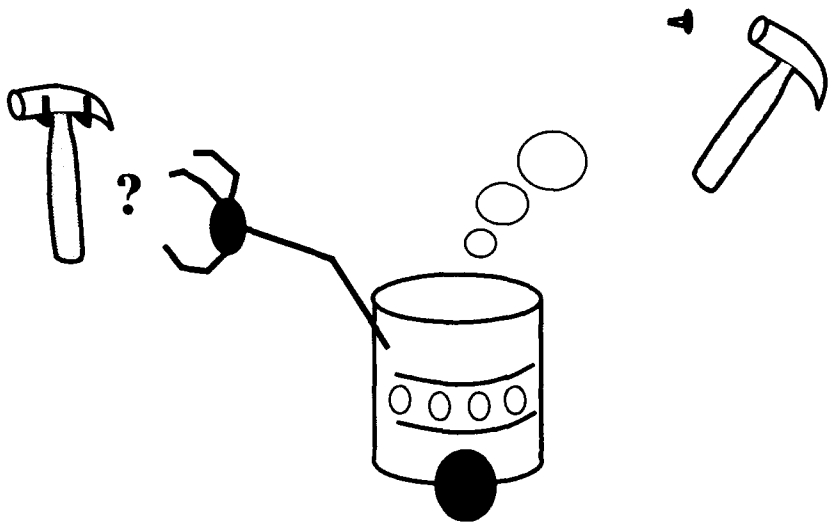


Figure 1.1: Given a task description, target object and environment geometry, and hand kinematics and geometry, the robot must figure out how to grasp the hammer.

supporting the hammer. We would know that we want to use the hammer to strike a nail. We would also have a model of the kinematics and geometry of the robot. Given all this information, the problem would be to synthesize a grasp of the hammer so that it can be used to strike a nail.

The simplest way to solve this problem might be to have a specialized hammer grasping routine. Once the location of the hammer is known, a specialized hammer grasping routine can be executed to grasp it. The routine would be designed to produce a grasp suitable for using the hammer to strike a nail.

Such a routine might have problems, however, if a non-standard hammer design was encountered, as in Figure 1.2, or if the environment was cluttered, as in Figure 1.3. Either situation may present difficulties not anticipated by the standard grasping routine. It may be difficult to determine how to best alter this routine so that the modified grasp satisfies the new constraints, yet continues to be suitable for the given task. There may in fact be no good grasps possible, and there may be no simple tests available to determine that this is the case.

In situations such as those just described, a more careful examination of the space of possible grasps is required. Unfortunately, this space is very large; it is exponential in the number of degrees of freedom of the robot. In addition, the constraints that must be satisfied—avoiding collisions, remaining within the kinematic limits of the hand, generating a grasp appropriate for the given task—are very different. There is no graceful way

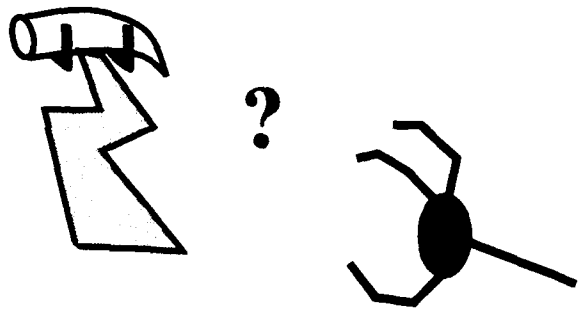


Figure 1.2: A non-standard tool design can make it difficult to use a grasping strategy tailored for that tool.

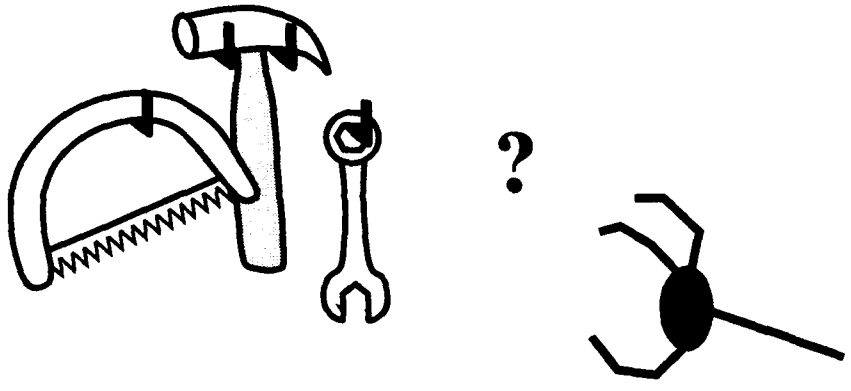


Figure 1.3: A cluttered environment can also make it difficult to use a standard grasping strategy.

to combine these constraints, and there are no easy ways to identify the space of good solutions.

One interesting option is to combine the standard grasp with a careful analysis of each problem situation. A canonical hammer grasp can still be used as a reference point in the search for a good grasp, but this canonical grasp can be generalized to cover a wider variety of situations. It is important that the generalization used retain the original objective of the hammer grasp—to use the hammer to strike a nail. It is also important that there be an efficient way to map such a generalized grasp onto a new situation in all of its complexity. This report explores this possibility.

1.2 Background

The first robot hands were simple grippers with a single degree of freedom: they could only open and close. The problem of grasping an object with this type of gripper could be described as the problem of maneuvering the gripper near the object and then positioning this gripper so that a stable grasp of the object would be produced when the gripper closed. Lozano-Pérez et al. [30], Pertin-Troccaz [38], and others [26] [22] [36] have developed systems that solve this grasping problem. They can move the robot to the object without producing a collision, grasp the object, set it down and regrasp it if necessary, and then place the object in a designated goal configuration.

Some work on the problem of reducing the uncertainty in the configuration of a part (as required for the design of an automatic parts feeder) is similar to the problem of synthesizing grasps for a simple gripper in the sense that a very limited repertoire of operations are applied to the target object. These operations may be repeated grasping by a single degree of freedom gripper as in Goldberg and Mason [15] and Brost [6], they may involve passing the part through a set of fences as in Peshkin and Sanderson [39], or they may involve tilting the table on which the part is resting as in Christiansen et al. [9] and Erdmann et al. [12]. As in the case of performing pick-and-place operations with a single degree of freedom gripper, the space of operations that may be performed on any given part is sufficiently small that in simple cases (when there is assumed to be no friction and no uncertainty in part modelling or mechanism control) the entire space of possibilities can be analyzed. When parts feeder design is less constrained as in Caine [7], however, this type of complete analysis is no longer practical. Instead, Caine demonstrates an interactive design approach that helps the designer to visualize and directly modify limiting features in a currently active parts feeder design.

As with design of a complex parts feeder, the problem of grasp synthesis for an articulated robot hand does not lend itself to a comprehensive analysis. An articulated hand can achieve a greater variety of grasps than a single degree of freedom gripper, but it is in general difficult to characterize the full range of possibilities. The complete search space is simply too large. This has led to a movement away from complete tasks such as performing a pick-and-place operation and toward more specialized subproblems. Two specialized subproblems that are important to this report are:

1. the selection of good points of contact on a target object, and

2. the development of tailored grasping strategies.

Work on these two subproblems is briefly reviewed here.

Finding Good Contact Sets

Before good points of contact can be found on a target object, some means for evaluating a proposed contact set is necessary. Work on the analysis of stable grasps can be found in Mishra et al. [32], Cutkosky [10], Kerr and Roth [23], Salisbury [42], and Salisbury and Craig [43]. Representative of work on optimization functions, or grasp quality measures are Li, Hsu, and Sastry [27], who measure the ability to manipulate a grasp object, and Kirkpatrick and Yap [24], who measure the ability to apply task forces and torques to a grasped object. Work in this area most relevant to this report is reviewed in detail in Chapter 2.

Once an optimization function has been specified, optimal sets of contacts can be found on a target object. In this area, Hanafusa and Asada [17] find a grasp of a two-dimensional object by minimizing the energy stored in springs at the contacts. Baker et al. [2] show that a stable grasp of a polygon can be formed by placing contacts at points of intersection of the polygon with the maximum radius circle that can fit inside it. Park and Starr [37] use a set of heuristics to optimize three-contact grasps of polygonal objects. Markenscoff and Papadimitriou [31] find three-contact grasps that minimize the sum of forces required to lift a two-dimensional polygon. Ponce et al. [41] use a different type of objective function, finding four-contact grasps of polyhedral objects that maximize the size of independent contact regions such that as long as each contact falls within its designated region, the grasp will be a good one. Li and Sastry [28] propose an objective function that minimizes the sum magnitude applied forces required to counter the worst case task wrench within a given task wrench space ellipsoid. Work in this area most relevant to this report is reviewed in detail in Chapter 3.

Developing Tailored Grasping Strategies

Work on developing grasping strategies for articulated hands is motivated by studies of human grasping. Napier [33] divides grasps into power and precision grasps. Jeannerod [21] studies the effect of sensor feedback on hand shapes and hand trajectories during prehension tasks. Klatsky and Lederman [25] explore the use of sensors in the hand to determine properties of objects. Iberall and MacKensie [19] contains a good overview of work in this area.

Representative of work in the category of developing tailored grasping strategies for robot hands is Bard and Troccaz [3], who decompose the target object into ellipsoids and propose a grasping strategy designed for grasping ellipsoidal objects. Cutkosky and Howe [11] construct a taxonomy of manufacturing grasps and develop an expert system to choose between them. Vuskovic and Marjanski [45] describe some grasping behaviors

aimed at reducing data input lines for prosthetics. Brock [5] develops tools for the construction and analysis of sensor-based grasping strategies that eliminate the need to know an object's exact configuration in the workspace. Work in this area is reviewed in detail in Chapter 5.

Both specialized approaches to grasp synthesis—finding optimal sets of contacts and developing tailored grasping strategies—have advantages and limitations. Although solution techniques that find optimal sets of contacts have the ability to globally optimize an objective function related to the utility of the resulting grasp, they impose many constraints on the problem space in order to efficiently achieve this goal. These constraints take the form of requiring either a very small or a very large number of contacts, and limiting the range of application of the solution to two-dimensional objects or to point contact on the faces of a polyhedron (see Sections 2.3 and 3.2). In addition, once a set of contacts is selected, some means of achieving this set of contacts must be found. It is not useful to find the globally optimal set of contacts if no configuration of the robot hand can possibly achieve these contacts.

The selection or development of a standard grasping strategy limits the capabilities of the robot hand to the point these capabilities can be fully analyzed, at least in simple cases, as was done with single degree of freedom grippers and some parts feeder problems. This allows the constraints of hand kinematics and geometry to be incorporated directly into the construction of a good grasp. Although this type of preplanned strategy can be very robust to some uncertainties, such as uncertainty in target object location (Brock [5]), the approach in general has not demonstrated much flexibility in the face of changes in the geometry of the target object, as in the non-standard tool design of Figure 1.2, or the addition of troubling obstacles, as in the cluttered environment shown in Figure 1.3. The flexibility of the hand allows adjustments to be made to the grasping strategy, but little work has been done on the problem of successfully making this kind of adjustment while ensuring that the objectives of the original grasping strategy are retained.

This report demonstrates a grasp synthesis technique that combines some of the advantages of the two specialized approaches described above. The solution begins with the selection of an example grasp, or grasp prototype, as in the second approach. The example grasp is used to reduce the search space of the problem. This example grasp is then generalized based on the task to be achieved. The space of grasps matching the example grasp is precisely defined, so that any grasp within this space is guaranteed to be suitable for the task to be performed. This space of matching grasps can be intersected with the space of grasps possible in a given problem situation, and an objective function related to the quality of the grasp can be optimized over the result, as in the first approach described above, where optimal contact sets were generated.

The various steps of this grasp synthesis technique require examining the problems of developing an objective function to measure grasp quality, finding high quality sets of contact points, and fitting the shape of the robot hand to the shape of the target object. Work related to each of these topics is reviewed in detail at the beginning of the first chapter to address the topic.

1.3 Problem Statement

The problem solved in this report can be described with the following specification:

Inputs:

- A geometric model of the target object,
- a geometric and kinematic model of a robot hand,
- a task,
- a geometric model of the environment, and
- an example grasp.

Problem:

- Describe the space of *collision-free grasps of the target object* that are similar to the example grasp and that are suitable for executing the given task.

This is illustrated in Figure 1.4 as a portion of a complete solution to the grasp synthesis problem. The solution described in this report is shown surrounded by a preprocessing step and a postprocessing step. From the left, constraints feed into all three steps. From the right, one additional input, the grasp library, feeds into the preprocessing step.

The preprocessing step examines the target object geometry and the task, and pulls from the grasp library an appropriate example grasp. The example grasp should be a grasp that is known to be effective for the given task. It is designed as a point from which to launch a search through the space of possible grasps. As we will see, use of the example grasp as described in this report drastically reduces the search space of the grasp synthesis problem.

One objective of the report is to explore the hypothesis that a very general example grasp can be successfully used in a wide variety of situations. For example, throughout the second half of the report, the robot hand is modelled as shown in Figure 1.5, and the very simple example grasp of a cylinder shown in two views in Figure 1.6 is used for a range of grasp synthesis problems. Grasps of complex objects, such as that shown in Figure 1.7 are identified, and grasps in cluttered situations, such as that shown in Figure 1.8 are also found, starting from the same cylindrical example grasp.

Even if an example grasp can be assumed to be widely applicable, selection of an appropriate example grasp is still a problem that must be addressed. This problem is discussed briefly in Section 8.1.4, but the construction and use of a library of grasp prototypes is left to future work.

The postprocessing step takes as input a description of the range of good hand configurations that both match the example grasp and are suitable for the task to be performed. It is important to have the information on good regions of solutions provided at this point

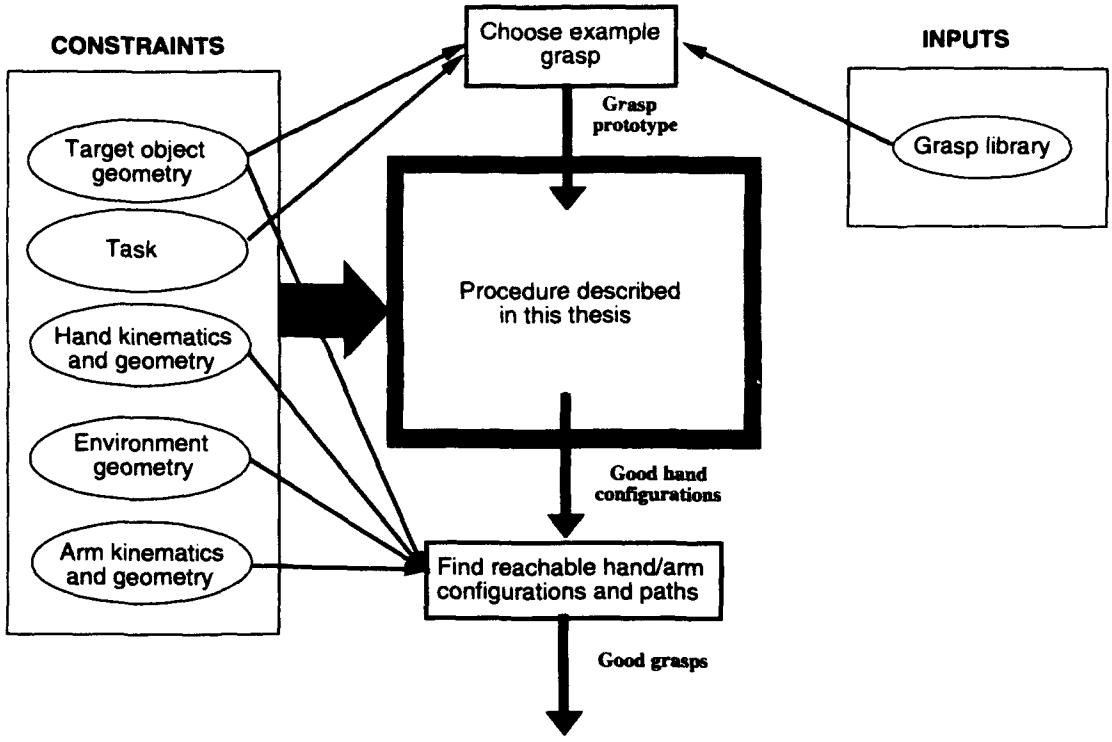


Figure 1.4: One solution to the grasp synthesis problem. The portion of the solution addressed by this report is framed by the black box. A complete solution has a preprocessing and a postprocessing step.

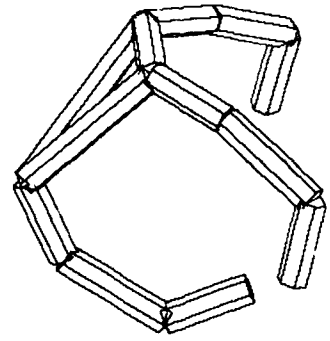


Figure 1.5: A simple model of the Salisbury hand. The hand has three fingers, each with three joints. The triangle represents the wrist.

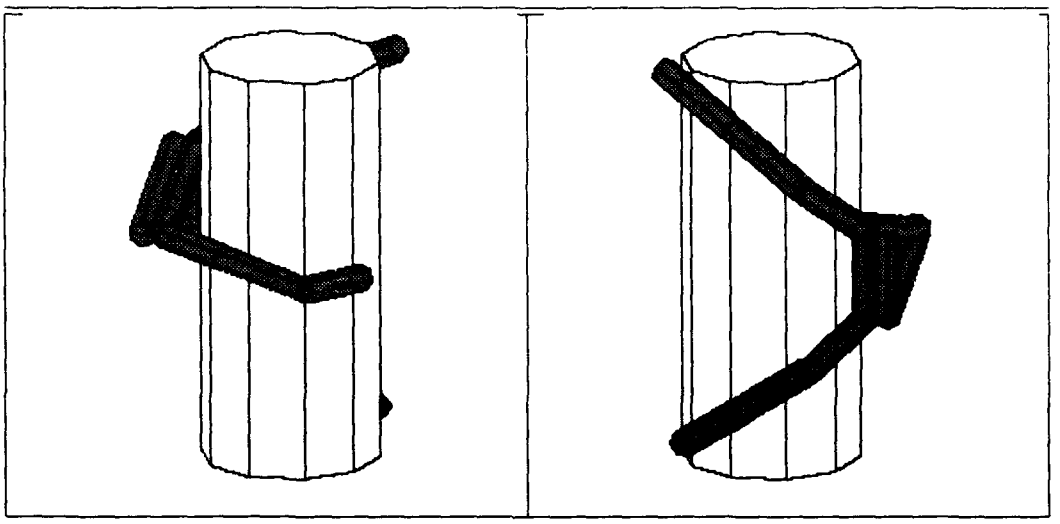


Figure 1.6: Two views of a prototype cylinder grasp.

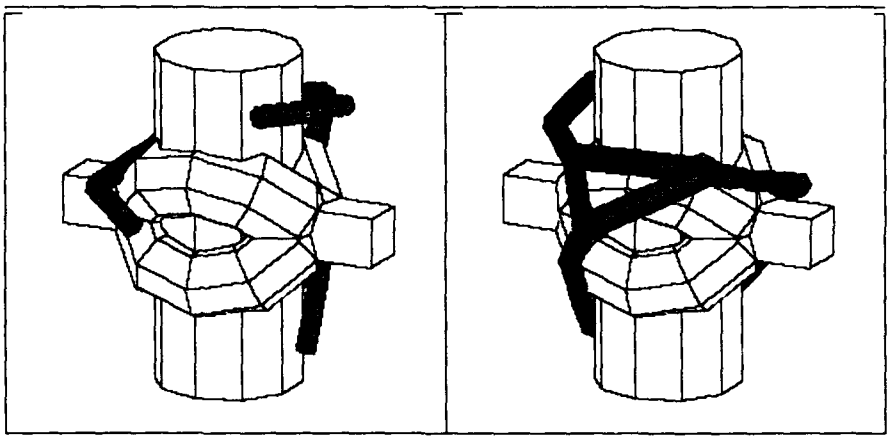


Figure 1.7: A match of the cylindrical example grasp to a more complex object.

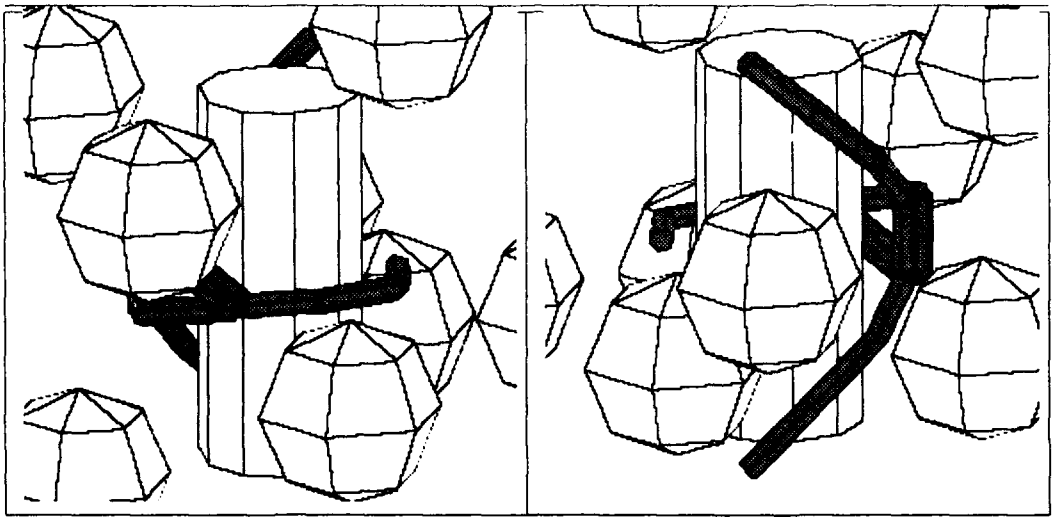


Figure 1.8: A match of the cylindrical example grasp to the cylindrical object in a crowded environment.

rather than choosing a single "optimal" solution, because there are more constraints that must be satisfied. In particular, it is necessary to find a final arm configuration that is free of collisions and to find a collision-free approach path into the final grasp. Figure 1.8 illustrates that it may not always be possible to find a free arm configuration or approach path for any given grasp that is selected. Although the grasp shown is free of collisions, it is unlikely that the hand could be maneuvered into that grasp by the robot arm unless some of the obstacles were removed. These last steps of the solution process are left to future work, as discussed in Section 8.1.

Providing regions of good hand configurations to the postprocessing step makes that step more feasible, but this region information is also useful in its own right. It can be used to compare the results obtained from using one example grasp to those obtained using a different example grasp. It can be used to quantify the effect of various obstacles on the space of good grasps that can be achieved. It can be used to select robust solutions, or grasps that are less sensitive to errors in modelling, sensing, and control, and it can be used to determine whether any grasps can be found at all. These uses are explored in the chapters that follow.

1.4 Approach

Figure 1.9 expands the diagram of Figure 1.4 to illustrate the approach to the grasp synthesis problem taken in this report. As described above, this approach assumes an input of a grasp prototype, or an example grasp, such as that shown in Figure 1.6. The output produced is a representation of the space of hand configurations forming collision-

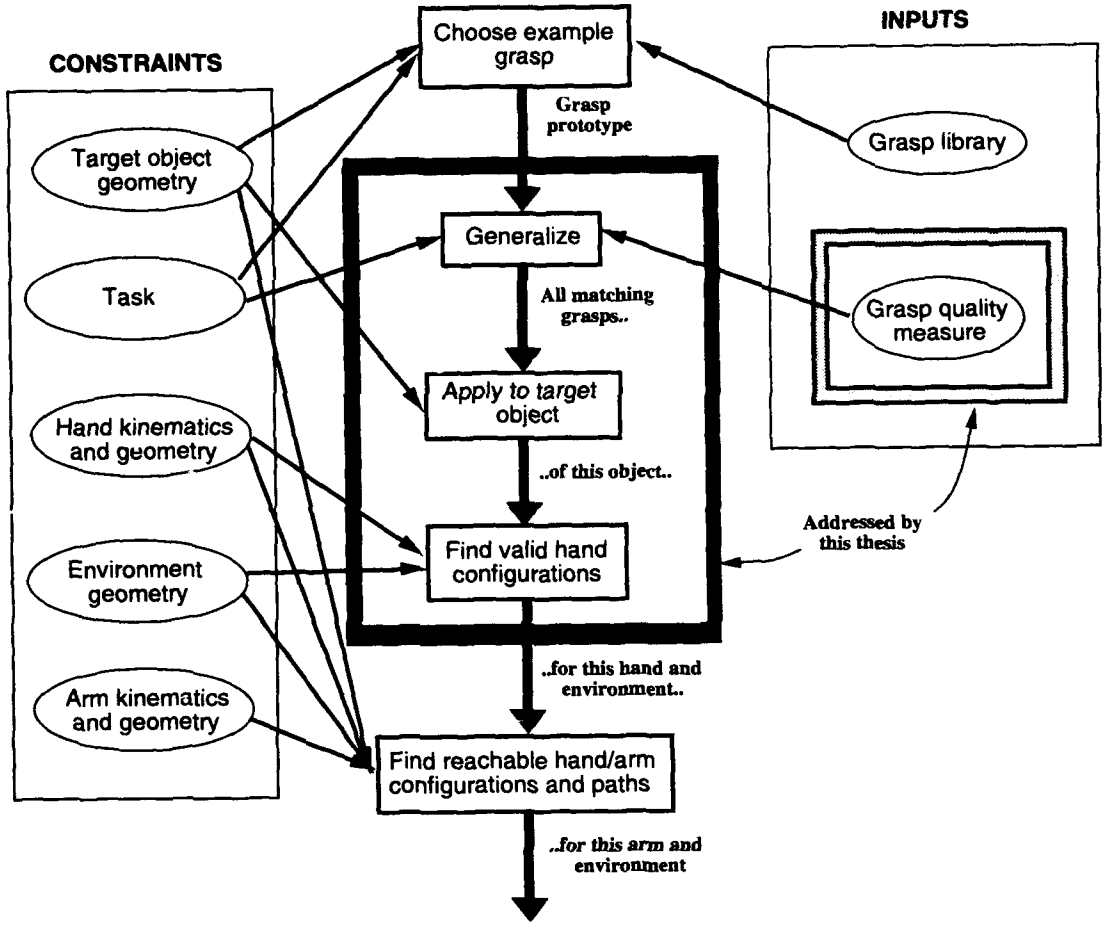


Figure 1.9: One solution to the grasp synthesis problem. The portions of the solution addressed by this report are framed by the black and gray boxes. A complete solution requires a preprocessing and a postprocessing step.

free grasps that match the example grasp and are suitable for the task to be performed.

The approach described in this report can be divided into three steps, with the inputs and outputs represented by icons in Figure 1.10. These steps are to generalize the example grasp, apply it to the target object, and incorporate the constraints of hand kinematics and environment geometry into the problem. In the first step of this process, contacts of the example grasp are extracted, and this set of contacts is generalized to produce a description of a range of contact sets appropriate for the given task. This step requires an additional input, the grasp quality measure, which must be kept above some threshold to ensure that a grasp is suitable for the given task. The region associated with each contact in the generalized grasp description is a convex portion of wrench space, as indicated by the plot icon in Figure 1.10. This means that the region associated with each contact can be efficiently described, and it can be efficiently projected onto the geometry of any target object.

In the second step of this process, the generalized grasp description is projected onto the geometry of the given target object. This results in sets of independent contact regions on the target object, such that as long as the robot hand makes some contact within each of the given regions, a good grasp can be formed. The icon in Figure 1.10 shows an example set of independent contact regions for a new target object.

In the third step of this process, the kinematics and geometry of the robot hand and the geometry of the environment are integrated into the solution. This step requires representing the range of collision-free hand configurations from which appropriate sets of contacts can be achieved. A parallel algorithm is developed to exploit the independence of the contact sets. The icon in Figure 1.10 shows one hand configuration that is free of collisions and matches the contact regions of the previous figure.

The chapters below work step by step through this solution. Chapter 2 begins by developing a set of tools that can be used to describe tasks and to define an appropriate grasp quality measure. Chapter 3 uses this grasp quality measure to construct equivalence classes of grasps based on the task and the example grasp. It shows how a given equivalence class of grasps can be projected onto a given target object geometry to obtain sets of independent contact regions. As long as the robot can achieve one contact within each of the given contact regions, a good grasp of the target object can be formed. Chapter 4 shows some examples of finding optimal sets of contact points using these concepts. This first half of the report uses two-dimensional object models for illustration, but the procedures developed apply to three-dimensional objects as well.

Chapter 5 shows how the constraints of the hand kinematics and geometry and the environment geometry can be added to the problem. The parallel algorithm described in this chapter is not very efficient, and Chapter 6 describes a variety of modifications and tradeoffs that must be made to make this algorithm work efficiently on today's parallel machines. Chapter 7 presents a variety of example grasp synthesis problems, describing additional practical concerns that arise in the context of these examples. Chapter 8 presents a summary and discussion. The second half of the report uses three-dimensional object models for illustration.

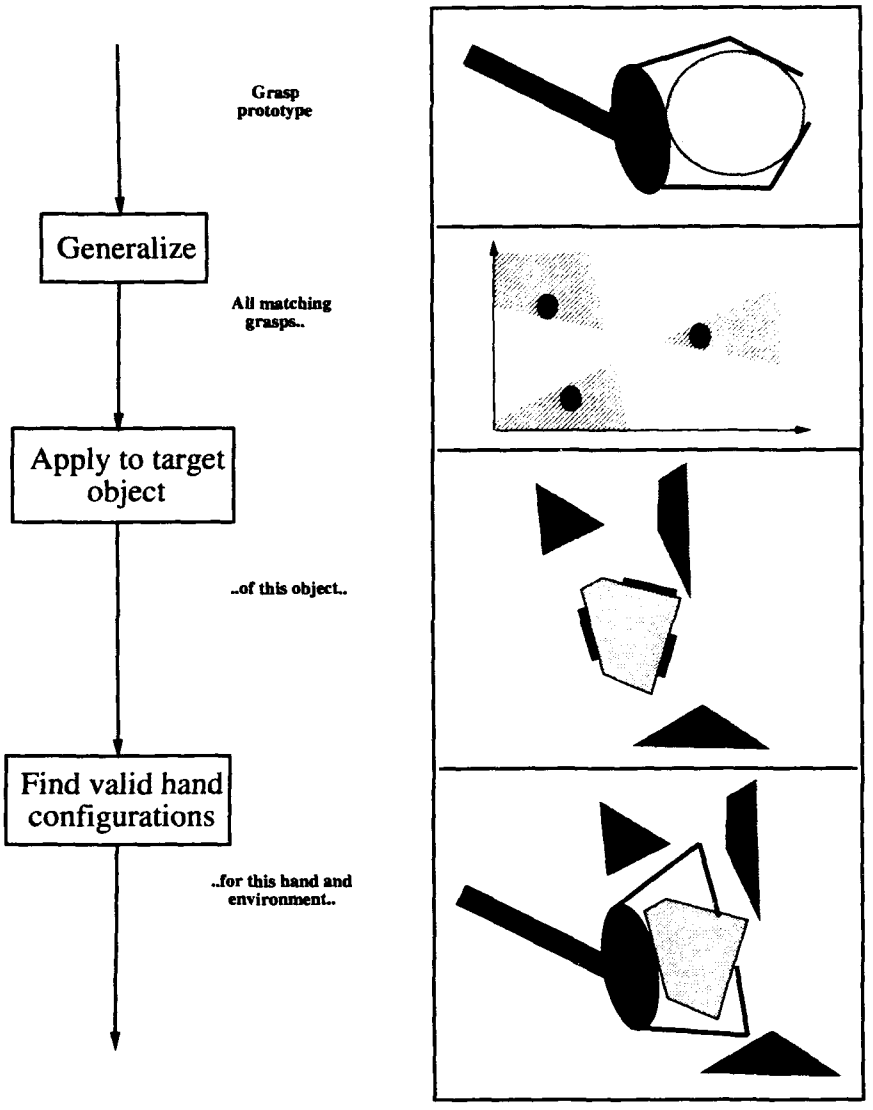


Figure 1.10: Grasp synthesis in this report is a three-step process that begins with an example grasp, defines a space of contact configurations that match the example grasp, intersects this description with the contact combinations possible for a given target object, and finds hand configurations that are free of collisions and can achieve a contact combination within the given set.

1.5 Contributions

The main contribution of this report is a system that:

- incorporates a variety of problem constraints, including target object geometry, environment geometry, hand kinematics and geometry, and a task, and
- produces a global solution: a space of wrist configurations from which high-quality, collision-free grasps are possible.

A global solution is useful, because it allows an estimate to be made of the robustness of a grasp to errors in wrist placement, it provides a quantitative measure of the effect of the obstacles in the environment on the space of possible grasps, and it indicates whether any solutions are possible. In addition, a global solution provides the flexibility needed to execute a postprocessing step, which must incorporate arm kinematics into the solution and find a collision-free path into one of the high-quality, collision-free grasps that have been identified.

One important component of the system described in this report is:

- a novel technique for generalizing standard grasps and applying them to new problems.

Standard grasps are used because they can drastically reduce the complexity of the search for a good grasp, but they can be very inflexible. The grasp generalization technique described in this report makes a standard grasp more effective by expanding its range of application to a wide variety of target object and environment geometries, while ensuring that the resulting grasps are appropriate for the intended task. Because this generalization technique allows any single standard grasp to apply to a wide range of problems, a small library of standard grasps is sufficient to cover a large space of possibilities.

A second important component of this system is:

- a novel technique for performing a parallel search through the space of hand configurations.

This technique takes advantage of the fact that the grasp generalization process allows the placement of each contact of the grasp to be independently optimized. This allows the process of assembling good hand configurations to be transformed into a dynamic programming algorithm. The many-dimensional space of hand configurations (this space has fifteen dimensions in the examples of this report) can be searched for optimal solutions by performing a small number of simple steps within a six-dimensional workspace.

Chapter 2

Measuring Grasp Quality

The main goal of this report is to present a general technique for grasp synthesis. We would like to synthesize only the best grasps, however, and to do this, we must carefully consider how a grasp should be evaluated.

Within the scope of this report, two properties are important for evaluating a grasp:

1. the suitability of a grasp for the task to be performed (Figure 2.1), and
2. the robustness of a grasp to errors in modelling, sensing, and control (Figure 2.2).

The first of these properties, the match of a grasp to a task, forms the basis for the *grasp quality measures* explored in this chapter. A task may require lifting the object, resisting external disturbance forces, and applying assembly forces. A high quality grasp will be capable of meeting these goals without the hand applying extreme forces to the object and without the object slipping from the grasp.

The second of these properties, the robustness of the grasp, is not considered in this chapter. Robustness is independent of the grasp quality measures developed in this chapter in the sense that it can be expressed in terms of these quality measures: a robust grasp can be described as a high quality grasp surrounded by a large, continuous region of high quality grasps. Because grasp robustness can be evaluated as a function of the grasp quality measures of this chapter, consideration of grasp robustness can be left to discussions in Chapters 3 and 4.

The goal of this chapter is to explore how the fit of a grasp to a task can be captured with a numerical value. The results of this exploration will allow us to determine, for example, that the top grasp of Figure 2.3 is much better than the bottom grasp of the same figure for the simple task of countering arbitrary disturbance forces. The ability to rank grasps in this way will enable us to synthesize grasps that are a good fit to given tasks in the chapters that follow.

It is important to note that although the pictures and examples of this chapter involve only two-dimensional objects, the techniques that are developed are general. The trivial extensions required to accommodate three-dimensional target objects are covered in Section 2.10.

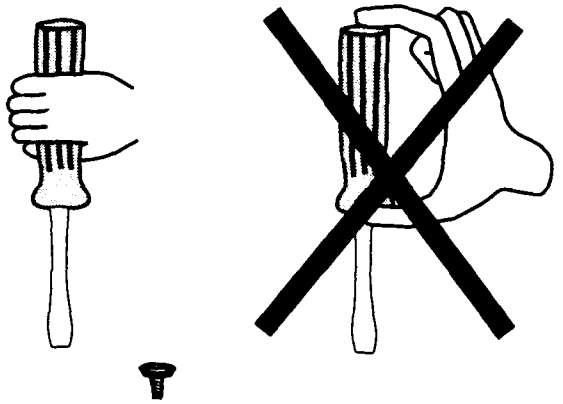


Figure 2.1: A grasp must be capable of applying appropriate task forces. This concern forms the basis for the grasp quality measures explored in this chapter.

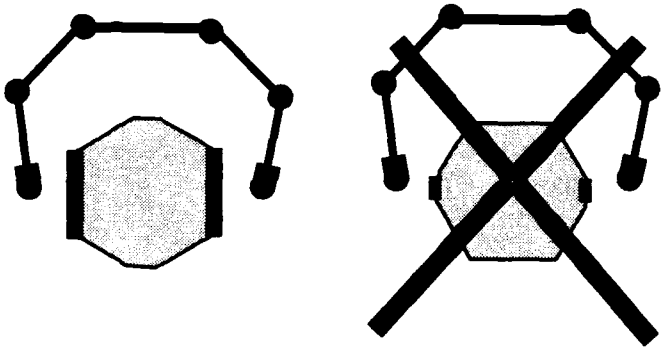
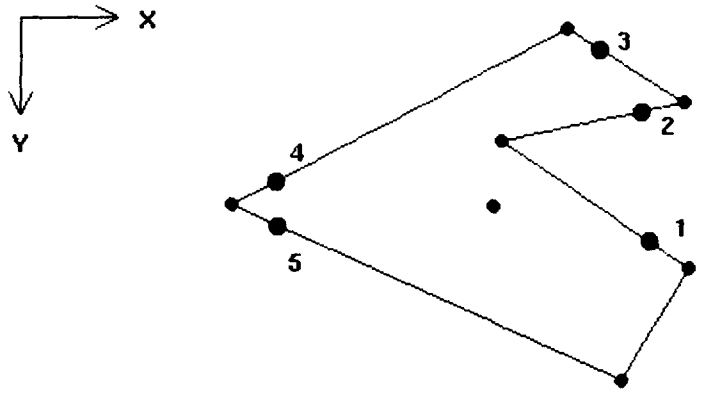
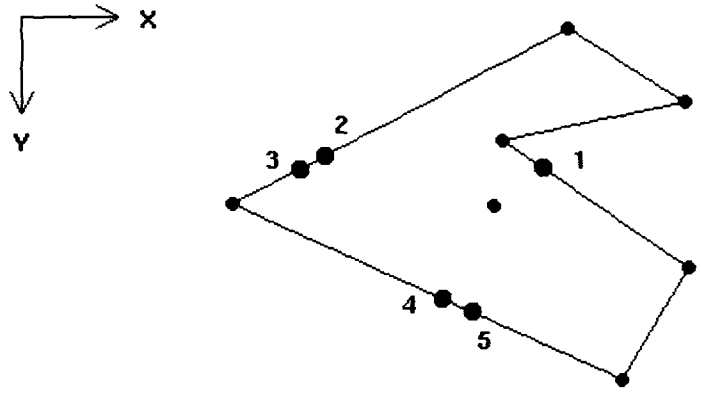


Figure 2.2: A grasp is more likely to be successful if acceptable areas of contact are large and robust to errors in modelling, sensing, and control. Grasp robustness is addressed later in the report.



Quality = 0.31



Quality = 0.00

Figure 2.3: This figure shows two grasps, each produced by applying forces at the number contact points. The grasp in the top figure, is much better than the marginal grasp in the bottom figure with respect to the ability to counter arbitrary disturbance forces applied to the polygon. The quality measure provides one indication that this is the case.

We also note that the simple and general grasp quality measure used throughout most of this report is not new (see Section 2.3). The analysis provided in this chapter is important, however, because it examines the assumptions behind this grasp quality measure, and because it provides general tools for constructing grasp quality measures for specific tasks.

The next few sections further describe the problem that will be solved. Section 2.1 gives a detailed outline of the problem and Section 2.2 outlines the notation used in this chapter. Section 2.3 reviews previous definitions of grasp quality measures, and Section 2.4 presents a brief overview of the body of the chapter, where tools for constructing grasp quality measures are developed.

2.1 Problem Statement

This chapter addresses the problem of selecting a quality measure that ranks the fit of a grasp to a given task. The problem specifications for this chapter are as follows:

Definitions:

- **A grasp:** a set of contacts on a target object.
- **A task:** defined by the *task wrench space*, or the space of resultant forces and torques that must be applied to the target object.

Inputs:

- A geometric model of the target object,
- a grasp,
- a task.

Assumptions:

- Frictionless point contacts,
- no singular contacts (Figure 2.4),
- no complex contacts (Figure 2.5),
- contact torques defined about the target object center of mass.

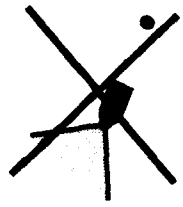


Figure 2.4: For contact with two-dimensional objects, point-vertex contact is considered too fragile to be part of a good grasp.

Figure 2.5: Because they do not have a unique local contact normal, complex contacts are not discussed in this chapter.

Problem:

- Estimate the suitability of the grasp for the given task.

The next few paragraphs elaborate upon these problem specifications. In particular, they describe how a grasp is specified, provide a means for measuring the capabilities of any given grasp, and describe how a task is specified. Although the pictures and examples of this chapter use two-dimensional objects, this is not listed as one of the assumptions above. The techniques that are developed throughout this chapter are general, and the examples in the second half of the report involve three-dimensional objects. Section 2.10 covers the trivial extensions needed to accommodate three-dimensional objects.

A Grasp is a Set of Contacts

This chapter considers a very simple description of a grasp. A geometric model of a target object is given, and a grasp of that target object is formed by placing some number of contacts on the object.

Forces and Torques can be Determined from the Contacts of a Grasp

The forces and torques that can be exerted on the target object through a grasp determine the capabilities of that grasp. The assumptions listed above make it very easy to

Figure 2.6: Non-singular, simple contact types for a two-dimensional object. These contacts all have unique local normals.

identify these forces and torques. First, it is assumed that all contacts are non-singular, simple point contacts. For two-dimensional objects, this includes only the contact types illustrated in Figure 2.6: edge contact at a vertex, point contact on an edge, and point contact on a curved surface. The examples of this chapter use only the first two of these contact types. The third is not used simply because curved objects are not examined.

Second, it is assumed that all contacts with the target object are frictionless. For the types of contacts that will be considered in this chapter, this means that the direction of the force applied to the target object at any contact can be determined uniquely from the local contact geometry. In the absence of friction, force can only be applied in the direction of the local contact normal, which can easily be calculated for the three contact types shown in Figure 2.6.

The torque exerted on the target object can be computed from the applied forces at the contacts, but first a point must be selected about which to measure the torque values. For convenience, torques are measured about the target object center of mass. This seems like a harmless assumption, but it does affect some of the grasp quality measures described in the sections below. This assumption is reviewed in Section 2.8, after the two grasp quality measures have been discussed in detail.

A Task is a Space of Forces and Torques

To determine the suitability of the grasp for a particular task, a description of that task is needed. In this report, a task is a space of forces and torques that the hand may be required to apply to the target object as it performs some function with that object. This space of forces and torques is referred to as the *task wrench space*. This chapter discusses the derivation of one specific task wrench space in some detail.

2.2 Notation

A number of terms are used in this chapter:

$\underline{w}_i$ = wrench due to applied force at contact $i = \begin{bmatrix} \underline{f}_i \\ \underline{\tau}_i \end{bmatrix}$

$\underline{f}_i$ = applied force at contact i

$\hat{\underline{n}}_i$ = unit outward pointing normal at contact i

$\underline{\tau}_i$ = scaled torque due to applied force at contact i, $= \lambda(\underline{f}_i \times \underline{d}_i)$

$\underline{d}_i$ = vector from contact point to object center of mass

λ = multiplier to convert torque values to units of force

n = number of contacts

α_i = multiplier for applied wrench at contact i

There are several things worth special notice in this list of terms. First, note that the wrench construct has been introduced. The term *wrench* comes from screw theory (see Hunt [18] for a discussion), which is used to analyze the kinematics of mechanisms. In this report, however, wrenches are just used to refer to a combined vector of forces and torques.

Second, note that torque multiplier λ is used to relate torque magnitudes to force magnitudes. In this report, torque multiplier λ is set deterministically for any given target object. It equals the reciprocal of the maximum magnitude torque that could possibly be applied to that target object using unit applied force, and it has units of $\frac{1}{length}$. This *method of weighting torques* depends on the worst case torque that could be exerted on an object, and it is one way of making the capabilities of a particular grasp independent of target object scale. The effect of this choice of λ can be summarized with the following expression:

$$|\underline{\tau}_i| \leq |\underline{f}_i|. \tag{2.1}$$

Third, notice that all torques $\underline{\tau}_i$ are defined about the center of mass of the target object (using vector $\underline{d}_i$), as stated in the assumptions of Section 2.1. This decision affects the grasp quality measures outlined, and it is reviewed in Section 2.8.

Fourth, note how the capabilities of a grasp are expressed using the parameters listed above. Legal values for applied forces $\underline{f}_i$ can be computed from the contacts of a grasp and the assumption of frictionless point contacts as described in the previous section. Torques $\underline{\tau}_i$ can be computed from contact points and applied forces. Contact wrenches can be formed from these forces and torques. Given suitably normalized contact wrenches $\underline{w}_i$ and suitable limitations on wrench multipliers α_i, the *grasp wrench space* can be expressed as the set of vectors that satisfy:

$$\sum_{i=1}^{n} \alpha_i \underline{w}_i. \tag{2.2}$$

for some set of legal α_i. Measuring grasp quality then becomes a problem of:

- choosing how to normalize contact wrenches $\underline{w}_i$,
- choosing limitations on wrench multipliers α_i, and
- determining how well the grasp wrench space fits a given task wrench space.

These issues will be discussed in the body of this chapter.

2.3 Previous Work

There has been a considerable amount of work directly related to the problem of selecting a grasp quality measure. Grasp quality measures in the literature have focused on one of three properties of a grasp:

- sensitivity to contact placement,
- ability to manipulate the grasped object, and
- ability to apply wrenches to the grasped object.

The paragraphs below briefly review work on the first two of these properties and then explore in more detail work on the third property, which is directly related to the grasp quality measure of this chapter.

2.3.1 Sensitivity to Contact Placement

The sensitivity of a grasp to errors in contact placement is one measure of the robustness of that grasp. A grasp that allows some freedom in positioning each contact will be resistant to small errors in modelling the object and small errors in controlling the hand. Robust grasps can be constructed by maximizing the sizes of contact regions on the object such that as long as each contact falls within its designated region, the grasp can be made stable. Using contact region size as a quality measure, Ponce et al. [41] show how to construct optimal four-contact grasps of polyhedral objects and Faverjon and Ponce [13] show how to construct optimal two-contact grasps of curved objects. Nguyen [35] describes a variety of robust contact configurations, including three-contact grasps of polyhedral objects. Section 3.6.7 of this report describes how a similar quality measure based on contact region size can be combined with the grasp quality measure described in this chapter, which evaluates the space of forces and torques that can be applied to an object.

2.3.2 Ability to Manipulate the Grasped Object

When a task requires the grasped object to be manipulated by the robot hand, grasp quality can be measured by determining how efficiently the hand can generate the required task motion. Li, Hsu, and Sastry [27] address this question, modelling the motion of the grasped object with a *manipulability ellipsoid.* They propose a grasp quality measure that estimates how well the incremental motions that can be applied to the object by the hand match the motions that are required for the task.

This report has the goal of generating fixed, stable grasps. The configuration of the object within the hand is not expected to change. A grasp quality measure based on ability to manipulate the object within the grasp is not needed for this type of grasp synthesis problem.

Reference	Objective Function	Task	Constraints
[8]	$\min \mu$	force closure	2 contact, 2D, curved object
[4]	$\min \mu$	force closure	2 contact, 2D, curved object
[20]	$\min \mu$	lift object	3 contact, 3D, polyhedron, hand in contact
[31]	$\min \sum_i \lvert \underline{f}_i \rvert$	lift object	3 contact, 2D, polygon
[31]	$\min \sum_i \lvert \underline{f}_i \rvert$	lift, rotate object	4 contact, 2D, convex polygon

Figure 2.7: Grasp quality measures used for grasp synthesis. The objective functions given measure the suitability of a grasp for the specified task. The last column indicates constraints on the solution proposed in each reference. Parameter μ represents the coefficient of friction at a contact with the target object.

Reference	Objective Function	Task
[24]	$\min \left(\max_i \lvert \underline{f}_i \rvert \right)$	task wrench space ball
[14]	$\min \sum_i \lvert \underline{f}_i \rvert$	task wrench space ball
[14]	$\min \left(\max_i \lvert \underline{f}_i \rvert \right)$	task wrench space ball
[28]	$\min \sum_i \lvert \underline{f}_i \rvert$	task wrench space ellipsoids

Figure 2.8: Other proposed grasp quality measures. The objective functions given measure the suitability of a grasp for the specified task.

2.3.3 Ability to Apply Wrenches to the Grasped Object

When a task requires a set of wrenches to be exerted on an object, grasp quality can be measured by determining how effectively the hand can generate these wrenches. This is the grasp quality measure considered in this chapter.

Ideally, this grasp quality measure is constructed using a two step process. First, the space of torques that can be applied to the joints of the robot hand is mapped to the space of wrenches that can be applied to the grasped object. Then, this is compared to the space of wrenches required for the task. A very common abstraction, however, which is also used in this chapter, is to construct the grasp quality measure from contact forces rather than from joint torques. This is useful because it results in a grasp quality measure that is independent of the device used to achieve the contact forces. All of the work described below uses this abstraction. The tables in Figures 2.7 and 2.8 summarize some important properties of the work described in this section.

If a grasp is described as a set of contacts and a task is described as a space of

wrenches, then the basic approach to measuring the quality of the grasp is to see how well the wrenches that can be applied to the object using legal combinations of forces at the contacts match the wrenches required for the task. The questions that must be answered when choosing such a grasp quality measure are how the task wrenches should be expressed and how the penalty function on applied wrenches should be measured.

One very simple task is to achieve a force closure grasp. A force closure grasp is capable of resisting at least some amount of external force in any direction, although the grasp may rely on friction to counter external forces in some directions. The wrench space of this task is an arbitrarily small ball centered at the wrench space origin. With the goal of achieving a force closure grasp, Chen and Burdick [8] minimize the size of the friction coefficient required to form a two-contact grasp of a two-dimensional curved object. Blake and Taylor [4] demonstrate a fast method for finding locally optimal two-contact grasps of two-dimensional curved objects under this same grasp quality measure. In related work, Trinkle [44] demonstrates a linear programming technique for determining whether a grasp is force closure either with or without dependence on friction and for any number of contacts.

Another common task is to lift the object. The wrench space of this task can be described as a single force vector through the object center of mass. With the goal of achieving this task, Jameson [20] performs local grasp optimization, minimizing the coefficient of friction required to lift the object using a three-contact grasp of a three-dimensional polyhedron. Jameson is simulating grasping using a robot hand, not just contact points, and so he adds to his grasp quality measure potential fields designed to keep contacts of the grasp away from the ends of the fingers, keep the joints of the hand from their limits, and keep the fingers of the hand away from the table. Markenscoff and Papadimitriou [31] minimize the sum magnitudes of the forces required to lift a two-dimensional object using a three-contact grasp of that object. The object weight is assumed to lie in the third dimension. Markenscoff and Papadimitriou also show how to minimize the worst case sum of the magnitudes of the contact forces of a four-contact grasp of a two-dimensional convex polygon such that the grasp can counter a unit force in any direction. This represents a task wrench space appropriate for lifting and slowly rotating the grasped object or for performing translational acceleration of the grasped object.

If the task wrenches are completely unknown, the task wrench space can be described as a ball centered at the wrench space origin. Kirkpatrick and Yap [24] propose to minimize the sum magnitude contact forces required to counter the worst case wrench within this task wrench space. Ferrari and Canny [14] develop this measure and propose a second alternative that minimizes the maximum force required to achieve the same task. Because the task wrench space now has a substantial torque component, both of these measures require that a weighting factor be defined to relate forces and torques.

There may be enough information about the task to favor certain directions of the task wrench space over others. Li and Sastry [28] propose the use of task wrench space ellipsoids, and they develop a grasp quality measure that minimizes the sum magnitude applied forces required to counter the worst case task wrench within a given wrench space ellipsoid. Grupen and Weiss [16] perform local grasp optimization, minimizing the contact

forces required to achieve wrenches along specified task basis vectors. These techniques also require a weighting factor relating forces and torques.

The grasp quality measure used in the grasp synthesis examples in this report is similar to those described in [24] and [14].

2.4 Chapter Overview

This chapter develops a set of tools for constructing grasp quality measures. The grasp quality measure has been abstracted away from the robot hand, and so a grasp has been defined as a set of contact points on a target object. One aim of this chapter is to show how this abstract definition of a grasp can be used to compute the grasp wrench space. Another goal is to determine exactly how a task wrench space might be defined. Once the grasp wrench space is known, and a task wrench space has been given, grasp quality can be defined as the ratio of the size of the grasp wrench space to the size of the task wrench space.

A number of different techniques for measuring grasp quality for specific tasks were shown in Section 2.3. Due to assumptions made in Chapter 3 of this report, which constrain how the grasp wrench space can be measured, and due to a desire to develop tools for use with more general task wrench spaces, a grasp quality measure cannot be arbitrarily chosen from Section 2.3. Section 2.5 covers the constraints imposed on the grasp wrench space measurement, and Section 2.6 presents an example grasp wrench space. There are fewer constraints on the task wrench space, and Section 2.7 presents one simple example. Section 2.8 uses the example task wrench space to illustrate construction of a grasp quality measure. This grasp quality measure proves difficult to compute, and so a simple approximation is proposed and evaluated. Section 2.9 addresses the issue of more complex tasks. The examples of this chapter all involve two-dimensional objects, and Section 2.10 covers the simple extensions required to form grasp quality measures for three-dimensional objects. Section 2.11 presents a summary.

2.5 Measuring the Grasp Wrench Space

This section describes the grasp wrench space construction used in this report. It first outlines the assumptions made in Chapter 3 that constrain this construction and then describes how the grasp wrench space can reasonably be measured within the bounds of these assumptions.

Chapter 3 covers the topic of forming equivalence classes of grasps from a single example grasp. These grasp equivalence classes are defined based on the grasp quality measure, and the constructions used in that chapter work with the convex hull of the contact wrenches of a grasp. In other words, Chapter 3 assumes:

- The convex hull of the contact wrenches of a grasp can be used to measure grasp quality.

Recall that the grasp wrench space was expressed as the space spanned by

$$\sum_{i=1}^{n} \alpha_i \underline{w}_i. \tag{2.3}$$

The assumption of Chapter 3 states that this space must be bounded by the convex hull of the contact wrenches. This can be achieved for a grasp wrench space of magnitude W with the following set of limits:

$$\alpha_i \geq 0, \tag{2.4}$$

$$\sum_{i=1}^{n} \alpha_i \leq W, \tag{2.5}$$

$$|\underline{f}_i| = 1. \tag{2.6}$$

Then, the unit grasp wrench space can be described as follows:

- The unit grasp wrench space is bounded by the convex hull of the contact wrenches formed from unit applied forces at the contacts of the grasp.

This answers two of the questions left open in Section 2.2: how $\underline{w}_i$ should be normalized, and how weights α_i should be limited. Note that in Expression 2.6, wrench magnitudes, rather than force magnitudes, could have been limited (using the expression $|\underline{w}_i| = 1$). Forces are limited rather than wrenches because the sum of the magnitudes of the applied forces of a grasp is a somewhat better predictor of the physical limitations of a hand than the sum of the magnitudes of the contact wrenches of the grasp. Contact wrench magnitude depends on parameters such as contact placement and target object geometry, which are related very indirectly to the physical limitations of the robot hand. It also depends on the position of the coordinate frame about which torques are measured, which is not at all related to the physical limits of the robot hand.

It is interesting to pursue the question of how the resulting description of the grasp wrench space does relate to the physical constraints of the robot hand. One type of analysis that can be performed is to roughly relate this grasp wrench space to the maximum torque that can be applied at any joint of the robot hand. This works as follows. Given the geometry of the hand, the maximum torque on any of the joints that may result from a unit force applied at a single contact somewhere on the hand can be estimated. From the worst case contact point and from the maximum torque that can be applied at each of the joints of the hand, an upper bound can be obtained on the magnitude of any single force that the finger can be guaranteed to resist. If there are multiple contacts on the same finger of the hand, then their effect on the joint torques required at that finger is additive. This results in an upper bound on the sum magnitude forces applied to that finger. The effects of forces on independent fingers should be independent, so limiting the sum magnitude of all applied forces results in a conservative estimate of the capabilities of the hand when this measure is related to individual joint torque limits.

2.6 An Example Grasp Wrench Space

The previous section presented a definition for a unit grasp wrench space. This section derives the unit grasp wrench space for an example grasp, showing how the general constraints on grasp wrench space construction, the constraints imposed by target object geometry, and the constraints imposed by a particular contact distribution progressively limit the grasp wrench space. The paragraphs below illustrate the effect of the various constraints by describing three wrench spaces:

- The *limit wrench space* is the wrench space bounded by the convex hull of the wrenches *that meet the general force and torque constraints of the problem*: $|f_i| = 1$ and $|\tau_i| \leq 1$.

- The *object wrench space* is the wrench space bounded by the convex hull of the wrenches that meet the general force and torque constraints of the problem *and can be applied to the target object.* The size and shape of this space are limited by the target object geometry.

- The *grasp wrench space* is the wrench space bounded by the convex hull of the wrenches that meet the general force and torque constraints of the problem *and form a given grasp of the target object.* This is the space of Equation 2.3. The size and shape of this space are limited by the placement of contacts on the target object boundary.

Note that the object wrench space of a target object is the union of all possible grasp wrench spaces of that object. The limit wrench space is the union of the object wrench spaces of all possible objects.

These three spaces are illustrated using an example five-contact grasp of a two-dimensional polygon. Figure 2.9 shows this grasp, with the contact points represented as dots in the figure. The paragraphs below derive the limit wrench space for the problem, the object wrench space of the example polygon, and grasp wrench space for the example grasp.

2.6.1 The Limit Wrench Space

The limit wrench space for a two-dimensional target object is constrained only by the limits placed on applied force in Equation 2.6 and the technique used for computing a weighted torque value from this applied force. These limits are reviewed by expanding the wrench space constraints on any grasp of a two-dimensional object:

$$\underline{w}_i = \begin{bmatrix} \underline{f}_i \\ \tau_{i,z} \end{bmatrix}, \tag{2.7}$$

$$\underline{f}_i = \begin{bmatrix} f_{i,x} \\ f_{i,y} \end{bmatrix}, \tag{2.8}$$

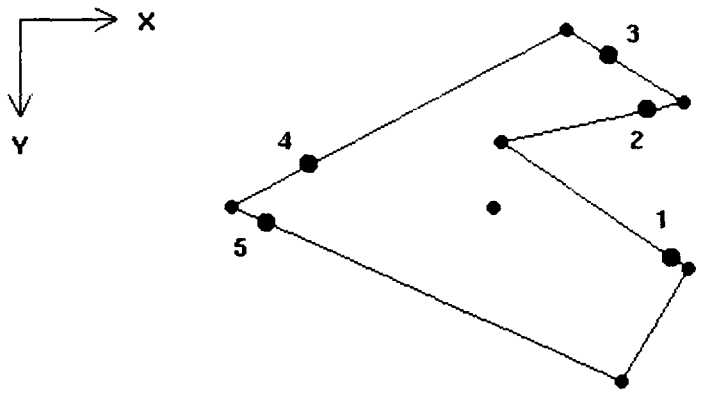


Figure 2.9: Grasp of a two-dimensional polygon, represented by forces applied at the numbered points.

subject to

$$|\underline{f}_i| = 1, \tag{2.9}$$

$$\tau_{i,z} = \lambda (\underline{f}_i \times \underline{d}_i) \cdot \hat{z}. \tag{2.10}$$

Every contact wrench $\underline{w}_i$ at a contact i is composed of a force component $\underline{f}_i$ of magnitude 1, and a torque component $\tau_{i,z}$. The torque component depends on the vector from the contact point to the target object center of mass ($\underline{d}_i$) and the torque multiplier λ. As in Section 2.2, λ is defined based on worst case torque in order to make grasps independent of target object scale. If X is the maximum torque arm possible for frictionless contact with the example polygon, then λ is defined as follows:

$$\lambda = \frac{1}{X}. \tag{2.11}$$

This guarantees that:

$$|\tau_{i,z}| \leq 1. \tag{2.12}$$

The unit limit wrench space of this problem is the space of wrenches $\underline{w}$ that meet the following conditions:

$$\underline{w} = \sum_{i=1}^{\infty} \alpha_i \underline{w}_i, \tag{2.13}$$

$$\alpha_i \geq 0, \tag{2.14}$$

$$\sum_{i=1}^{\infty} \alpha_i \leq 1, \tag{2.15}$$

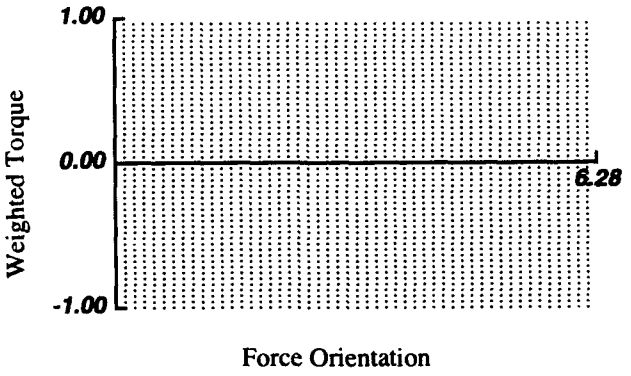


Figure 2.10: The set of normalized contact wrenches that satisfy the limit wrench space equations for the two-dimensional grasp synthesis problem fills a two-dimensional region. These wrenches can have any force orientation and any torque with magnitude less than or equal to 1.

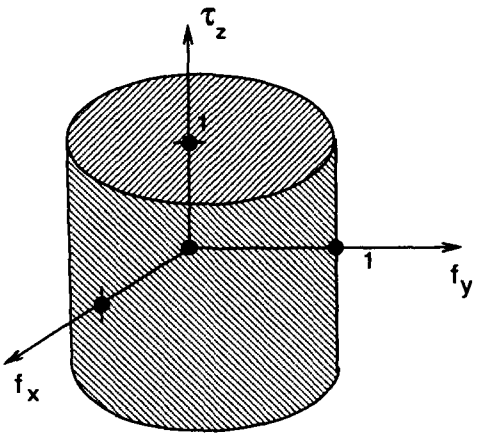


Figure 2.11: The limit wrench space for a planar problem is a cylinder in the three-dimensional wrench space of the planar problem. The boundary of this cylinder is represented by the plot in Figure 2.10.

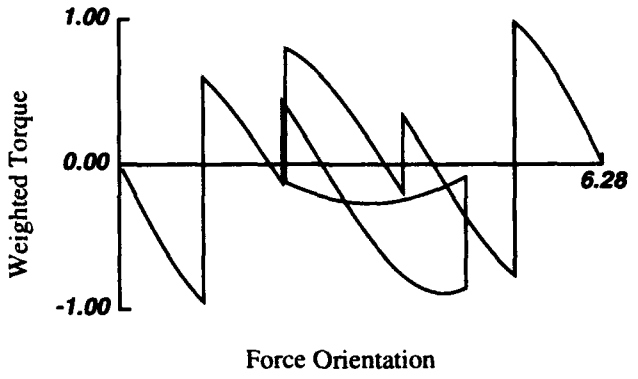


Figure 2.12: Curve traced by the perimeter of the example polygon. Edges are represented by vertical lines, because force orientation does not change with placement of a contact on an edge, although the size of the torque arm does change.

where $\underline{w}_i$ is *any contact wrench that satisfies Equations 2.9 and 2.12.*

Any wrench $\underline{w}_i$ that satisfies Equations 2.9 and 2.12 will lie on the curved surface of a cylinder. This cylinder surface can be unrolled and displayed as in the two-dimensional plot of Figure 2.10. The *unit limit* wrench space of Equation 2.13 is bounded by the convex hull of the wrenches indicated in Figure 2.10. This makes it the complete cylinder volume shown in Figure 2.11.

2.6.2 The Object Wrench Space

The object wrench space represents the best grasp of a target object that can ever be achieved. The unit object wrench space is a subset of the unit limit wrench space, and is derived by adding the additional condition that each wrench $\underline{w}_i$ is *a contact wrench due to some contact with the target object.*

The contact wrenches that result from contact with the example target object of Figure 2.9 cover only a portion of the cylinder surface in Figure 2.10. The boundary of the unit object wrench space is captured by tracking unit forces around the object perimeter. This traces out a curve on the surface of the unit cylinder. Figure 2.12 shows the object boundary curve for the example polygon. The vertical lines represent point contacts on edges of the polygon. On an edge, force direction remains constant while torque varies with motion along the object perimeter. If ds represents infinitesimal motion along an edge, then infinitesimal changes in torque are given by:

$$d\tau = \lambda \, ds. \tag{2.16}$$

The curved lines represent edge contacts at vertices of the polygon. At a vertex, force and torque vary as the contacting edge rotates. If $\underline{d}_v$ is the vector from a vertex to the

object center of mass and $\hat{\underline{f}}_v(\theta)$ is the unit contact force vector, we can write:

$$d\tau = -\lambda \left(\underline{d}_v \cdot \hat{\underline{f}}_v(\theta)\right) d\theta. \tag{2.17}$$

The convex hull bounding the unit object wrench space of the example polygon is constructed from the object boundary curve displayed in Figure 2.12. To display this convex hull, the curve is sampled and a convex hull is constructed from the sampled points. This sampled representation of the object wrench space of the example polygon is shown in Figure 2.13. The figure shows four views of the object wrench space. The wire frame cylinder can be used as a reference. This represents the corresponding limit wrench space boundary. The coordinate frame at the top of this cylinder can help relate the views to one another.

The features of the object boundary curve of Figure 2.12 can be seen in the views of this polyhedron. The view in the lower right-hand corner of Figure 2.13 shows the long, isolated curve that wraps around from approximately $\frac{5\pi}{3}$ radians through 2π or 0 radians to approximately $\frac{\pi}{3}$ radians in Figure 2.12. The view directly above that shows the muddled region centered about π radians.

The object wrench space has a very definite character. Large portions of the limit wrench space are not covered, and only certain types of grasps will be possible. If the objective in choosing a grasp is to cover the largest amount of the object wrench space possible, then the selection of logical contacts will be limited. For example, it will in general be good to select contacts at torque extremes and along the curved sections representing vertex contacts. This topic will be discussed further below.

2.6.3 The Grasp Wrench Space

Although the limit wrench space describes the volume within which the grasp wrench space must exist, and although the target object geometry limits the space of grasps that can be achieved, the real character of the grasp wrench space is determined by the choice of contact points that make up a grasp. For a set of n contact points, the unit grasp wrench space is the space of wrenches $\underline{w}$ that meet the following conditions:

$$\underline{w} = \sum_{i=1}^{n} \alpha_i \underline{w}_i, \tag{2.18}$$

$$\alpha_i \geq 0, \tag{2.19}$$

$$\sum_{i=1}^{n} \alpha_i \leq 1. \tag{2.20}$$

where $\underline{w}_i$ is *any contact wrench that can be applied to the target object at a contact point of the grasp and that satisfies Equations 2.9 and 2.12.*

The contact wrenches of the grasp shown in Figure 2.9 cover only a few points on the object boundary curve of Figure 2.12. These points are shown in Figure 2.14. This means that the grasp wrench space, which is formed from the convex hull of these points, will

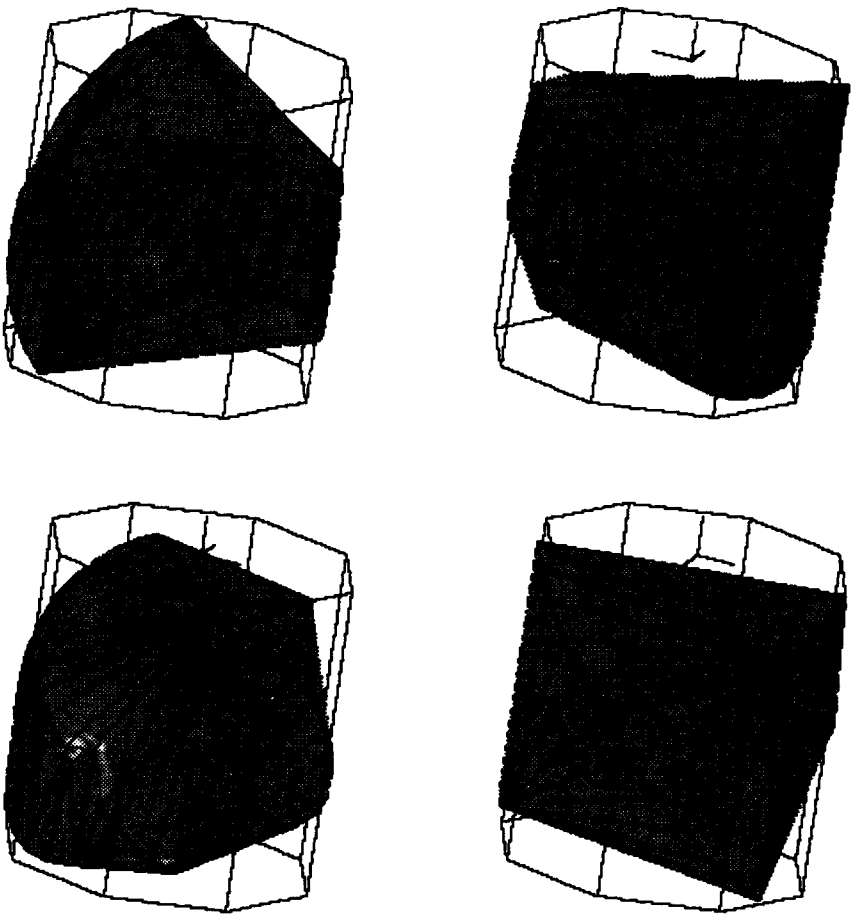


Figure 2.13: The object wrench space of the example polygon. The entire object wrench space falls within the limit space cylinder, and the boundary of the object wrench space polyhedron is formed from the convex hull of the points on the object boundary curve shown in Figure 2.12.

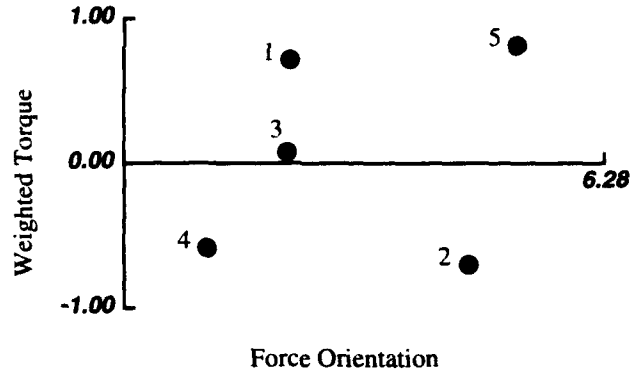


Figure 2.14: Unit contact wrenches of the example grasp.

be a small subset of the object wrench space. The grasp wrench space for the example grasp is shown in Figure 2.15. Compare this to the object wrench space, shown in Figure 2.13. Because the contact points are not distributed very evenly along the axis of force orientation in Figure 2.14, it can be seen in Figure 2.15 that the grasp wrench space is much wider when viewed along the x-axis (the right hand views of Figure 2.15) than when viewed along the y-axis.

2.7 An Example Task Wrench Space

The previous sections described how the grasp wrench space is constructed and measured in this report. The unit grasp wrench space was described as the wrench space bounded by the convex hull of the contact wrenches resulting from unit forces applied at the contacts of a grasp. A grasp quality measure, however, requires that both a grasp wrench space and a task wrench space be specified. This section defines an example task wrench space. This task wrench space is used to describe how grasp quality can be measured in the sections that follow.

The task wrench space developed in this section is based on the following goal:

- To estimate how effectively a grasp can counter arbitrary disturbance forces.

This is one way to measure the security of a grasp when little is known about the wrenches that will be encountered during execution of a task.

To develop a task wrench space for the task of countering arbitrary disturbance forces means to determine how these disturbance forces can be modelled. For this example, it is assumed that these forces follow the same rules as the contact forces of the grasp. In other words, the disturbances are forces applied to the target object, and they result only from non-singular, frictionless point contacts. This type of contact might result, for example, during contact between rigid bodies with low coefficients of friction in an

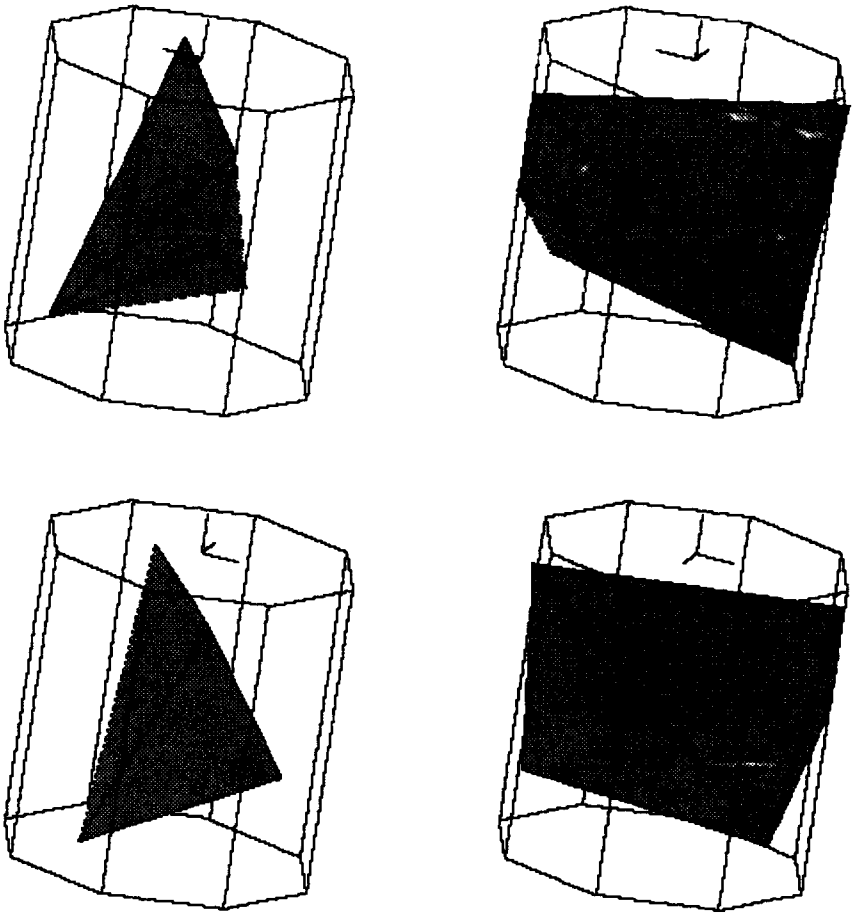


Figure 2.15: The grasp wrench space of a grasp of the example polygon. This is only a portion of the object wrench space shown in Figure 2.13.

assembly task. For disturbances that result from contact at multiple points, the sum magnitude of the disturbance forces applied to the target object is measured. This assumes that the disturbance forces are best described using an aggregate measure rather than a measure dependent on the number of contacts.

This technique for categorizing disturbance forces does not specifically capture disturbances that do not result from frictionless point contacts on the surface of the target object. This includes, for example, the wrench due to the target object weight, the wrench due to acceleration of the grasped object, or the resultant wrench due to a liquid contained by the target object. These wrenches may, however, be within the task wrench space as defined. If not, they can easily be added as described in Section 3.6.5.

Now that the rules for measuring a task wrench space have been specified, we can ask what shape this task wrench space will take. Because it is assumed that disturbance forces follow the same rules as the applied forces of the grasp, it is clear that a disturbance of magnitude less than or equal to one must fall somewhere within the unit object wrench space of the target object. A grasp that is expected to counter a disturbance force of magnitude one as defined in this section must therefore be prepared to counter any wrench within the unit object wrench space of the target object without losing the grasp. This space is used as the task wrench space in one definition of a grasp quality measure in the next section.

2.8 Two Grasp Quality Measures

The previous sections have shown how the grasp wrench space is measured in this report, and they have presented an example task wrench space designed to represent the simple task of countering arbitrary disturbance forces applied to a target object. This section considers the problem of measuring grasp quality, or the fit of a task wrench space to a grasp wrench space.

For this section and the remainder of the report, it is assumed that a task wrench space is given. A grasp must be capable of countering all wrenches within the task wrench space, and so a measure of grasp quality can be stated as follows:

- Grasp quality for a task is the reciprocal of the sum magnitude applied forces required for the task wrench space to just fit within the grasp wrench space.

A grasp quality measure will exist whenever a grasp wrench space spans the full range of directions spanned by the task wrench space. A special case exists when the grasp wrench space completely contains the wrench space origin. When the grasp wrench space contains the origin, then if the robot hand squeezes hard enough, a grasp can be formed that can successfully counter all wrenches within *any* task wrench space. The applied force magnitudes may be unnecessarily large, however, if the grasp wrench space bears little resemblance to the task wrench space, and this is reflected in a small grasp quality measure. Grasps can be compared based on the grasp quality measure for a given task. The relative effectiveness of a grasp for executing different tasks can also be evaluated by comparing the grasp quality measures of the grasp for the different tasks.

2.8.1 Measuring Grasp Quality for the Example Task

The next few paragraphs describe how a grasp quality measure can be formed for the example task of the previous section. In this task, the task wrench space was described as the unit object wrench space of the target object.

The grasp quality measure is the reciprocal of the sum magnitude applied forces of a grasp required for the grasp wrench space to just contain the entire task wrench space. Because the unit object wrench space is used as the task wrench space, it is helpful to scale both the task wrench space and the grasp wrench space by the reciprocal of the grasp quality measure and express this grasp quality measure as follows:

- $GQ1$: the scale of the largest object wrench space of a target object that fits entirely within the unit grasp wrench space of a grasp.

Grasp quality measure $GQ1$ can be estimated as follows:

1. For any n-contact grasp, construct convex hull H, which bounds the unit grasp wrench space. For the five-contact grasp of Figure 2.9, this convex hull is displayed in Figure 2.15:
$$H = CH\{\underline{w}_i : i = 1, \ldots, n\}. \tag{2.21}$$
H is composed of facets G of one dimension less than H:
$$G = \{g : g \text{ is a facet of } H\}.$$

2. Define the following parameters for each facet g:
$$\begin{array}{lcl} \hat{\underline{n}}_g & = & \text{unit outward-pointing normal of facet } g, \\ d_g & = & \text{distance of facet } g \text{ from the origin (positive in the } \hat{\underline{n}}_g \text{ direction).} \end{array}$$

3. Select m points to represent the boundary of the unit task wrench space. For this example, these could be the points of the object wrench space polyhedron in Figure 2.13:
$$P = \{\underline{p}_j : j = 1, \ldots, m\}. \tag{2.22}$$

4. Then the following equation represents an estimate of grasp quality:
$$e = \left[\max_{g \in G} \left(\max_{\underline{p}_j \in P} \frac{(\hat{\underline{n}}_g \cdot \underline{p}_j)}{d_g} \right) \right]^{-1}. \tag{2.23}$$

Term $\frac{(\hat{\underline{n}}_g \cdot \underline{p}_j)}{d_g}$ is the scale factor on the unit grasp wrench space required to place point $\underline{p}_j$ on the inner half space of facet g of the grasp wrench space convex hull. If this term is maximized over all points $\underline{p}_j$ and over all facets g, then the result is the factor by which the unit grasp wrench space must be scaled so that it contains the unit object wrench space. As described by the definition of grasp quality, the reciprocal of this term is the grasp

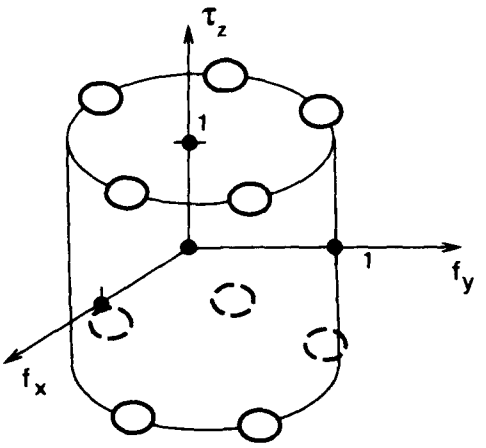


Figure 2.16: Considering only the limit wrench space of the polygon, we observe that the a good grasp might consist of evenly distributed forces at the torque extremes.

quality estimate e. If all points $\underline{p}_j$ of the task wrench space are scaled by factor e, then the scaled task wrench space will fit entirely within the unit grasp wrench space. Parameter e is an estimate of grasp quality measure $GQ1$ that is as good as the point approximation of the object wrench space represented by set P. Using the object wrench space shown in Figure 2.13, a grasp quality measure of 0.33 is obtained for the five-contact grasp wrench space shown in Figure 2.15. If all points of the polyhedron in Figure 2.13 are scaled by a factor of 0.33, the scaled polyhedron will fit within the polyhedron of Figure 2.15.

We can ask what a grasp quality measure of 0.33 means. For this example task, it means that a set of frictionless, point contact disturbance forces can be applied to the target object such that the sum magnitude of the grasp forces required to counter the disturbance forces is 3 times the sum magnitude of the disturbance forces.

We can also ask what properties high quality grasps will have. Looking just at the limit wrench space cylinder, we note that a high quality grasp would have a number of contacts evenly distributed in the space of force orientations and at both torque extremes (Figure 2.16). We can try to approximate this effect by picking off contacts from the object wrench space that fall at torque extremes.

One way to do this is to place contacts at the endpoints of each edge of the example polygon. Figure 2.17 shows the grasp wrench space of this 12-contact grasp. Compare this to the object wrench space in Figure 2.13. Using this object wrench space, a grasp quality measure of 0.44 is obtained. This is slightly better than the grasp quality measure of the 5-contact grasp.

Vertex contacts are needed to fill out this grasp wrench space. Figure 2.18 shows the grasp wrench space of the 18-contact grasp formed by adding a single contact at each vertex. This contact is oriented to bisect the range of orientations possible. Again using

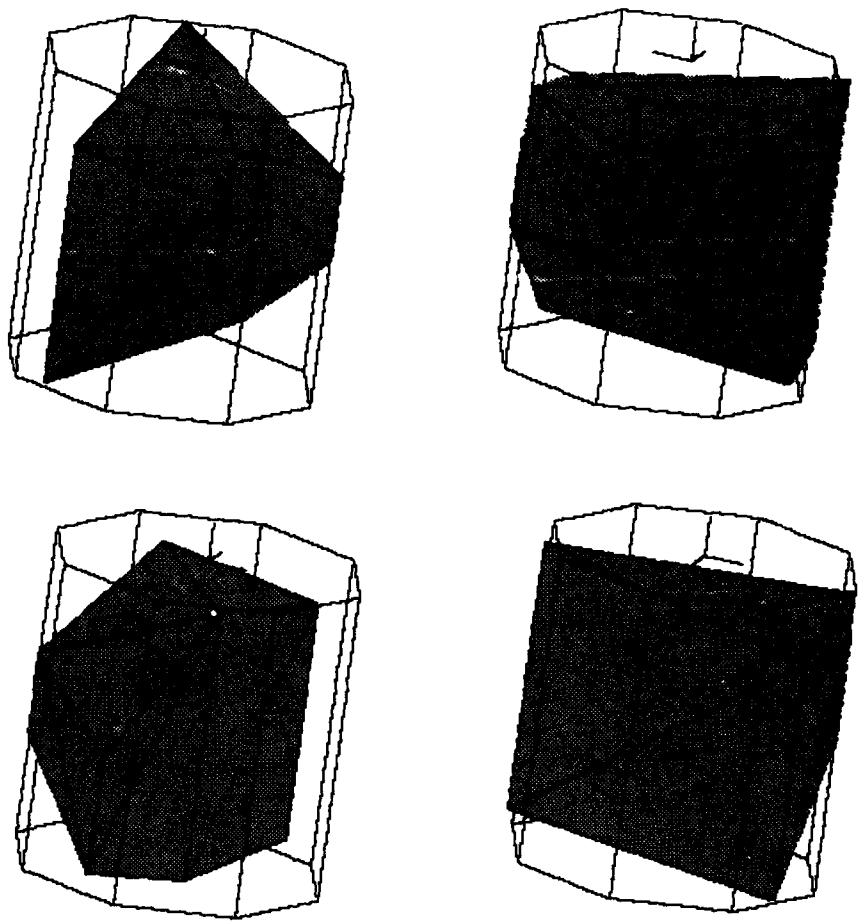


Figure 2.17: Wrench space covered by frictionless point contacts at the extreme points of each edge. This grasp has a grasp quality measure (GQ1) of 0.44.

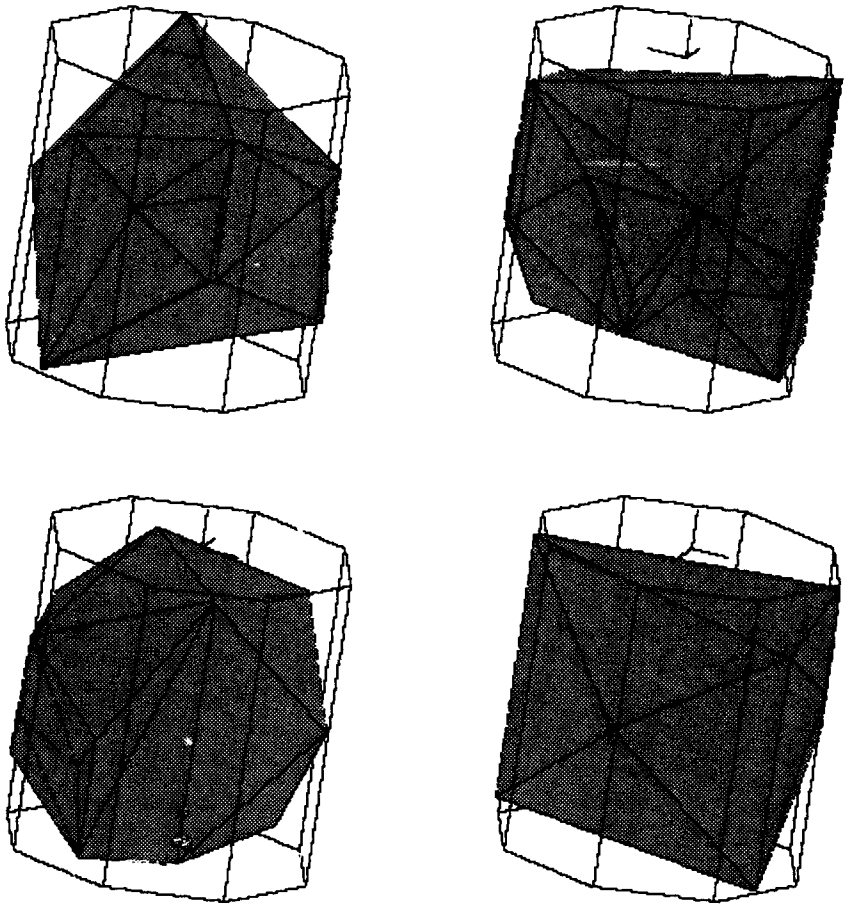


Figure 2.18: Wrench space covered by frictionless point contacts at the extreme points of each edge and one frictionless edge contact at each vertex. This grasp has a grasp quality measure (GQ1) of 0.83.

the object wrench space of Figure 2.13, the much higher grasp quality estimate of 0.83 is obtained.

2.8.2 Approximating Grasp Quality for the Example Task

Calculating an accurate grasp quality measure as described above will be quite time-consuming for problems of a realistic size. This is a result of the complexity incurred when using the object wrench space of the target object as the task wrench space. This section considers the question of how the grasp quality measure $GQ1$ might be approximated.

If we want a measure that is extremely simple to calculate and completely independent of target object complexity, there are two obvious options: a wrench space cylinder or a wrench space ball. The first option is derived by eliminating the constraints related to the target object geometry. With this option, we approximate the object wrench space very conservatively with the limit wrench space cylinder.

The second option is derived by beginning with the limit wrench space cylinder and eliminating the torque constraint, the constraint first specified in Equation 2.1. This allows a unit disturbance force to be associated with an arbitrarily large torque. Then task wrenches as a whole must be limited (e.g. $|\underline{w}_i| = 1$), rather than only the force component of these wrenches, to ensure that all disturbance torques are bounded. This results in an approximation of the object wrench space as a wrench space ball.

The *cylindrical approximation* would be a good one for target objects that have a nearly cylindrical object wrench space. This is the case when the object geometry contains many torque extremes of approximately the same magnitude. This is true, for example, of any regular polygon with more than a few edges. In general, though, the wrench space cylinder will be an overly conservative approximation of the object wrench space shape, and grasps will be designed to protect against many disturbance wrenches that are not possible.

The spherical approximation is less conservative in that the unit sphere will contain smaller regions of disturbance wrenches that are not possible, but this approximation also has problems. It does not capture any of the torque extremes of the object wrench space, and it is possible that a grasp with a high quality measure using this task wrench space will not protect against disturbance forces applied at these torque extremes. The next chapter shows that by searching only for grasps that match a given example grasp, or prototype grasp, we ensure that these torque extremes are often captured and that the wrench space ball provides a good estimate of the grasp quality measure $GQ1$.

Based on the preceding discussion, we propose to use the following grasp quality measure as an approximation of $GQ1$:

- $GQ2$: the size of the largest wrench space ball that fits completely within the unit grasp wrench space.

This is the grasp quality measure proposed in [24] and [14].

To compute the grasp quality measure $GQ2$ for an n-contact grasp of an object:

1. Compute the convex hull of the grasp:

$$H = CH\{\underline{w}_i : i = 1, \ldots, n\}. \tag{2.24}$$

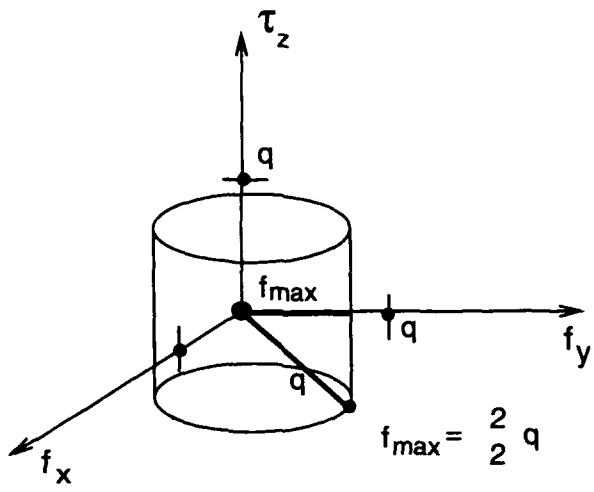


Figure 2.19: Grasp quality measure GQ1 can be compared to grasp quality approximation GQ2 (q).

H is bounded by the set of facets G:

$$G = \{g : g \text{ is a facet of } H\}.$$

2. Define the following vectors associated with each facet g:

$$\begin{aligned} \underline{\hat{n}}_g &= \text{unit outward pointing normal of facet } g, \\ \underline{p}_g &= \text{vector from wrench space origin to an arbitrary point on } g. \end{aligned}$$

3. Then quality measure q, which is the minimum perpendicular distance from a facet to the wrench space origin, can be computed as follows:

$$q = \min_{g \in G} (\underline{\hat{n}}_g \cdot \underline{p}_g). \tag{2.25}$$

2.8.3 Comparing the Grasp Quality Measures

It is possible to compare grasp quality measure $GQ1$ with approximation $GQ2$. In particular, given a $GQ2$ grasp quality measure q, a lower bound can be placed on the corresponding $GQ1$ grasp quality measure e. This lower bound is generated by fitting the limit wrench space cylinder into the grasp quality ball (Figure 2.19). This results in the lower bound:

$$e \geq \frac{\sqrt{2}}{2} q. \tag{2.26}$$

Grasp	q ($GQ2$)	e ($GQ1$)	Worst Case e ($\frac{\sqrt{2}}{2}q$)
5-Contact Grasp	0.35	0.33	0.25
12-Contact Grasp	0.44	0.44	0.31
18-Contact Grasp	0.80	0.83	0.56

Figure 2.20: Grasp quality approximation $GQ2$ vs. grasp quality measure $GQ1$ for the grasps seen so far.

We observe that by using the actual object wrench space rather than the limit cylinder, this bound cannot be improved. Because of the definition of weighted torque $\tau_{i,z}$, we know that at least one point in the object wrench space of size $\frac{\sqrt{2}}{2}q$ will have a force of magnitude $\frac{\sqrt{2}}{2}q$ and a torque $\tau_{i,z} \in \{\frac{\sqrt{2}}{2}q, -\frac{\sqrt{2}}{2}q\}$. This point will fall exactly on the quality ball of radius q. Increasing the size of the object wrench space will push this point outside the quality ball. Without knowing more about the grasp wrench space, we cannot determine whether e is in fact greater than $\frac{\sqrt{2}}{2}q$.

In practice, however, a grasp is not completely limited by the ball representing grasp quality. Quality measure e will generally be much better than $\frac{\sqrt{2}}{2}q$. In fact, e may even be greater than q, because the object wrench space of size q does not in general contain the entire wrench space ball of radius q. All of the grasps seen so far capture the extremes of the object wrench space sufficiently well for the grasp quality approximation represented by parameter q to be a very good predictor of the grasp quality measure e that was originally proposed. Figure 2.20 summarizes the $GQ2$ and $GQ1$ grasp quality measures for the 5-contact grasp in Figure 2.15, the 12-contact grasp in Figure 2.17, and the 18-contact grasp in Figure 2.18. The worst-case $GQ1$ quality measure, $\frac{\sqrt{2}}{2}q$, is included in the table for reference.

It is not surprising that the grasp quality approximation $GQ2$ agrees so well with the more intuitive grasp quality measure $GQ1$ for the examples presented in Figure 2.20, because these grasps were designed to capture the object wrench space of this object. It is more interesting to examine how useful this approximation will be when a grasp that was designed for one object is used to find a similar grasp on an object of different geometry, as in the next few chapters.

Figure 2.21 shows a set of grasps all designed based on the grasp of Figure 2.9. These are results from Chapter 3. The information used from the grasp of Figure 2.9 is very general, and so the "matching" grasps may appear very different from the original grasp. The details behind this matching process are covered in Chapter 3.

The objects in Figure 2.21 are arranged in order of decreasing similarity to the object for which the grasp was originally designed. The table in Figure 2.22 shows the comparison of $GQ2$ to $GQ1$ in these examples. From this table, we see that the approximate grasp quality measure $GQ2$ is a very good predictor of the more complex grasp quality measure $GQ1$ for all of these grasps, except perhaps in the case of object 3. This is not surprising.

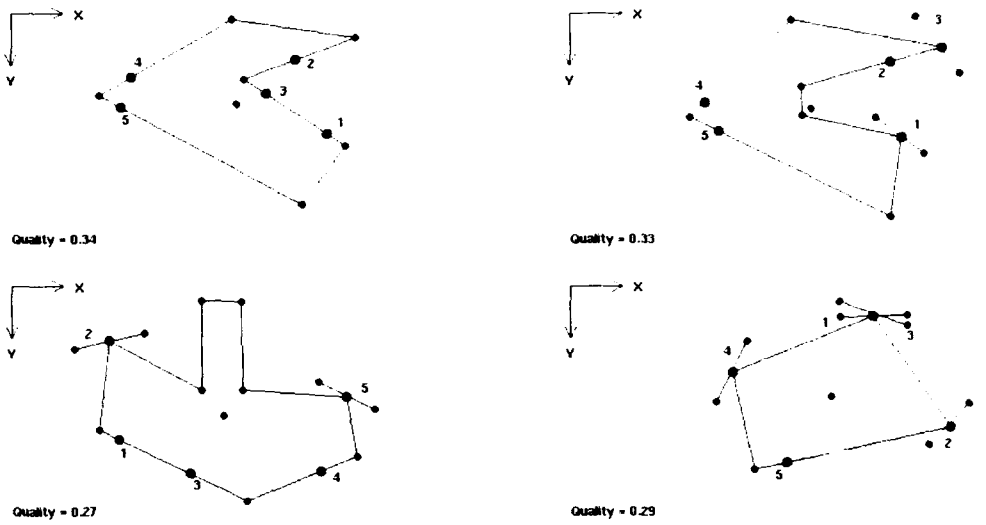


Figure 2.21: Grasps of four new objects (matching the example grasp) (UL) Grasp of object 1, (UR) Grasp of object 2, (LL) Grasp of object 3, (LR) Grasp of object 4.

Grasp	q $(GQ2)$	e $(GQ1)$	Worst Case e $(\frac{\sqrt{2}}{2}q)$
Grasp 1	0.34	0.31	0.24
Grasp 2	0.33	0.33	0.23
Grasp 3	0.27	0.22	0.19
Grasp 4	0.29	0.28	0.21

Figure 2.22: Grasp quality approximation GQ2 vs. grasp quality measure GQ1 for the grasps of the new objects.

Object 3 is the most complex object, and it does not look much like the original object. The grasp designed for the object in Figure 2.9 does *not* do a very good job of capturing the shape of the object wrench space of object 3.

2.8.4 Grasp Center Placement

At this point, the validity of the assumption presented in Section 2.1 that placed the grasp center at the target object center of mass can be examined. That section hinted that this decision would have an impact on the grasp quality values that would be measured. First, we note that this is not true of the grasp quality measure originally proposed, $GQ1$. This grasp quality measure compares the grasp wrench space of a given grasp of a target object to the object wrench space of that same object. As long as torques are measured about the same point when computing the two spaces, grasp quality will not vary with the placement of that point. In other words, the magnitude of the worst case disturbance force that can be resisted by a grasp cannot be affected by changing the reference frame in which torques are measured.

When the grasp quality approximation $GQ2$ is used, however, a different result is obtained. Changing the way in which torques are measured does affect the validity of the approximation of the object wrench space of an object as a wrench space ball. As the reference point about which torques are measured is moved, some parts of the object wrench space are compressed and others are expanded. This can cause the object wrench space to look less and less ball-shaped. The following justifications can be stated for choosing to measure torques about the target object center of mass:

- A well-balanced distribution of contact wrenches that can be applied to a target object is obtained, causing the object wrench space of the target object to be somewhat ball-shaped.

- The sensitivity of torque values to small changes in target object geometry is limited, allowing for a more meaningful comparison of grasp quality measures.

2.9 Grasp Quality Measures for more Complex Tasks

Every task, no matter how simple, is associated with some specialized task wrenches. One good example of a simple task with a known, narrow range of task wrenches is the task of drinking from a glass. The grasp must be capable of countering the weight of the glass and the weight of the liquid through the range of motion required for this task.

Specialized task wrenches give the task wrench space a more interesting shape, and the equations for estimating grasp quality are not as easily accessible as Equation 2.25. Section 3.6.5 shows that this is not a problem. The grasp synthesis techniques proposed in that chapter can easily be adjusted to accommodate any task wrench space. We will see that if the task wrench space does not depend on the target object geometry, even very complex tasks can be accommodated with no extra work required at runtime.

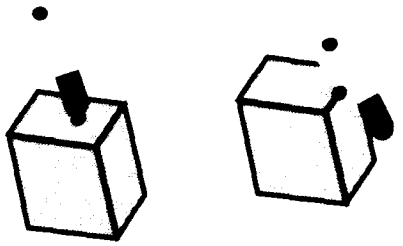


Figure 2.23: For grasps of three-dimensional objects, point contact on faces and edge contact on edges will be used. Local normals for these contact types can be easily computed.

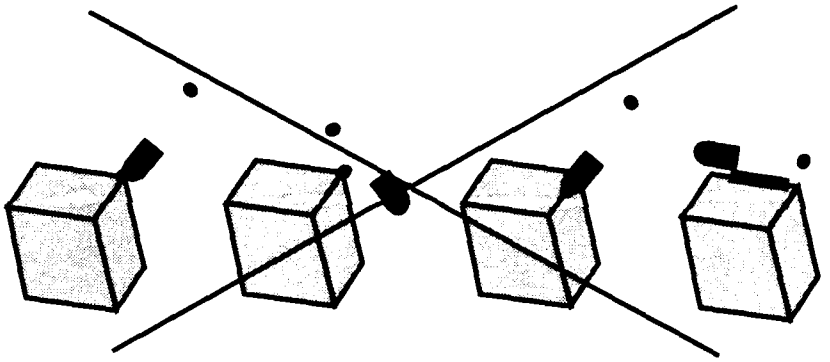


Figure 2.24: Point-vertex, edge-vertex, point-edge, and edge-edge contacts are considered too fragile to be part of a good grasp, unless the polyhedron vertex or edge is concave.

2.10 Working with 3D Objects

The main difference seen when moving to three dimensional objects is that there are different contact types. Figure 2.23 shows some of the contact types that are used in the three-dimensional examples of the second half of this report. Figure 2.24 shows the contact types that are considered to be singular, or not sufficiently robust to be part of a good grasp. For each of the contact types in Figure 2.23, a local normal can be computed at each contact point, which means that a contact wrench can be computed for every contact. Note that contacts in concavities, such as point contact in a hole or on a concave edge, are not singular in the same way as the contacts in Figure 2.24. Frictionless contact in a concavity can result in a range of contact wrenches, however. Section 8.1.2 discusses how this opportunity is exploited.

For grasps of three-dimensional objects, the wrench space has six dimensions: three

force dimensions and three torque dimensions. Each wrench is normalized, as before, so that

$$|\underline{f}_i| = 1, \tag{2.27}$$

and torque multiplier λ is defined to ensure that

$$|\underline{\tau}_i| \leq 1. \tag{2.28}$$

This gives the limit wrench space the shape of a three-dimensional unit ball when projected onto either force space or torque space.

From the contact wrenches of a grasp and from a given task wrench space, a grasp quality measure can be computed as described in the sections above. For example, Equations 2.23 and 2.25 can be used to compute grasp quality measures e ($GQ1$) and q ($GQ2$) respectively. The bound used to compare these two measures ($e \geq \frac{\sqrt{2}}{2}q$) also holds in this six-dimensional wrench space. The argument is similar. Because of the definition of weighted torque $\underline{\tau}_i$, we know that at least one point in the object wrench space of size $\frac{\sqrt{2}}{2}q$ will have a force of magnitude $\frac{\sqrt{2}}{2}q$ and a torque of magnitude $\frac{\sqrt{2}}{2}q$. This is the maximum magnitude wrench in the object wrench space of size $\frac{\sqrt{2}}{2}q$, and it has a magnitude of q. In other words, the object wrench space of size $\frac{\sqrt{2}}{2}q$ fits entirely within the wrench space ball of radius q, and in fact, one point of this space falls on the surface of this ball. We know then that e is at least $\frac{\sqrt{2}}{2}q$, but without more information on the grasp wrench space, we cannot determine whether e is greater than $\frac{\sqrt{2}}{2}q$.

2.11 Summary

This chapter introduced some techniques for constructing grasp quality measures. The grasp quality measures considered here characterized the fit of a grasp to a given task. This fit was measured by comparing the grasp wrench space, or the space of wrenches that could be applied to the target object, to the task wrench space, or the space of wrenches required to execute the given task.

Two grasp quality measures were described. The first was designed to represent the task of countering arbitrary disturbance forces that might be applied to the grasped object. This grasp quality measure proved difficult to compute and, not surprisingly, was highly dependent on target object geometry. A simple approximate grasp quality measure was then derived. This measure was chosen to be independent of target object geometry, and it was shown to be a good predictor of the more complex measure. This second, simpler grasp quality measure is used throughout the chapters that follow.

Chapter 3

Optimizing Contact Placement

The previous chapter addressed the problem of measuring grasp quality, and developed some tools for designing a grasp quality measure from a task description. The discussion in that chapter was simplified by viewing a grasp abstractly, as a set of contacts on a target object. This simplification was used to generate grasp quality measures independent of the robot hand.

This chapter continues to take advantage of this abstract definition of a grasp, pursuing the question of how to design such a grasp. The design process consists of finding a placement of contacts on a target object that produces a grasp with a high quality measure. This problem is interesting because of its complexity. If the goal is to design a grasp having more than two or three contacts, then the problem space will be much too large to perform a complete search. It is necessary to develop a means of constraining the space of grasps that will be examined, while ensuring that much of the space of interesting solutions is captured.

This chapter presents a novel technique for limiting the search space of this grasp synthesis problem. It uses knowledge that might be compiled from grasping many objects or from carefully analyzing a particular set of objects. This knowledge is compiled into a *grasp prototype*. The grasp prototype is defined in Section 3.1, and the body of this chapter describes how it can be matched to a target object to construct high quality grasps.

The technique for matching a grasp prototype to a target object is the main contribution of the chapter. This technique results in a grasp synthesis algorithm with the following properties:

- a match of the grasp prototype to the target object is defined based on the grasp quality measure,
- the quality measure of the grasp prototype is preserved,
- the search space of the grasp synthesis problem is drastically reduced,
- prototype complexity is unconstrained,
- target object geometry is unconstrained.

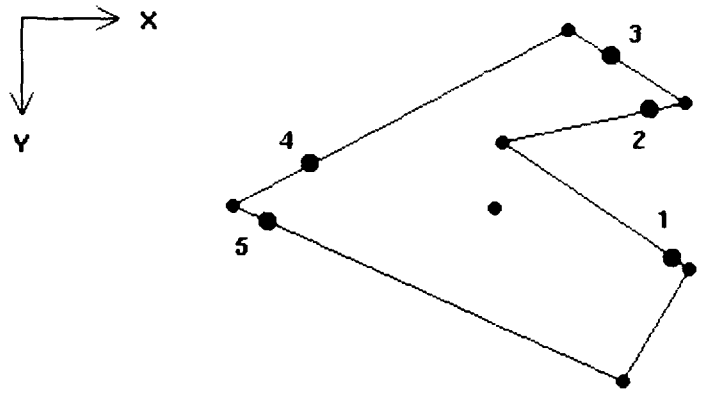


Figure 3.1: A two-dimensional prototype grasp, represented by forces applied at the numbered contact points.

All of these properties are examined in the sections that follow, but the most important of these properties are the first three. A match of the grasp prototype to the target object is defined in terms of the grasp quality measure and the features of the grasp — contact forces and torques — that contribute to the grasp quality measure. Because of this, it will be possible to generate grasps of the target object that are very different in appearance from the grasp prototype, but preserve the quality of that prototype. This provides a substantial amount of flexibility without a large cost in either speed of the algorithm or quality of the grasp.

The type of solution that will be obtained is illustrated in Figures 3.1 through 3.3. Figure 3.1 shows a two-dimensional prototype grasp. Figure 3.2 shows a matching grasp for an object very much like the prototype object, and Figure 3.3 shows a matching grasp for an object that is very different in appearance. Note that the contact ordering is different between the prototype grasp in Figure 3.1 and the grasp of the different object in Figure 3.3. Even in the grasp of the similar object in Figure 3.2, contact 3 has moved from its original position. These differences are one [illegible]ication that the prototype grasp is used in a flexible way. This chapter describes how this is done.

The examples in this chapter involve two-dimensional objects like these. The main purpose of the chapter is to illustrate the grasp synthesis algorithm, and two-dimensional examples are easy to visualize. The algorithm has been implemented for three-dimensional objects, but the three-dimensional examples are deferred to Chapter 7.

Section 3.1 begins the discussion by outlining the specifics of the grasp synthesis problem considered in this chapter. This problem is popular in the literature, and Section 3.2 reviews some previous work on similarly defined problems of grasp synthesis. Section 3.3

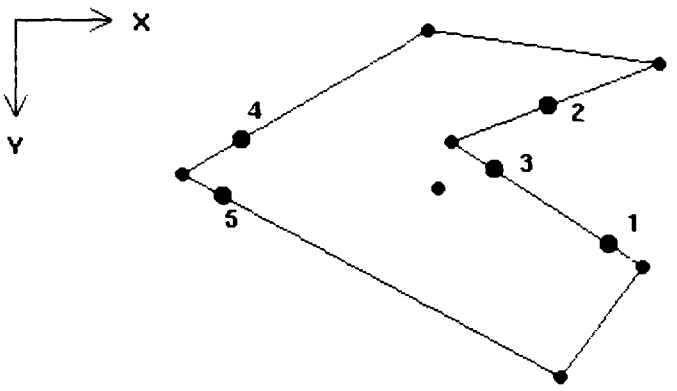


Figure 3.2: A matching grasp of a similar object. The most visible change is in the placement of contact 5.

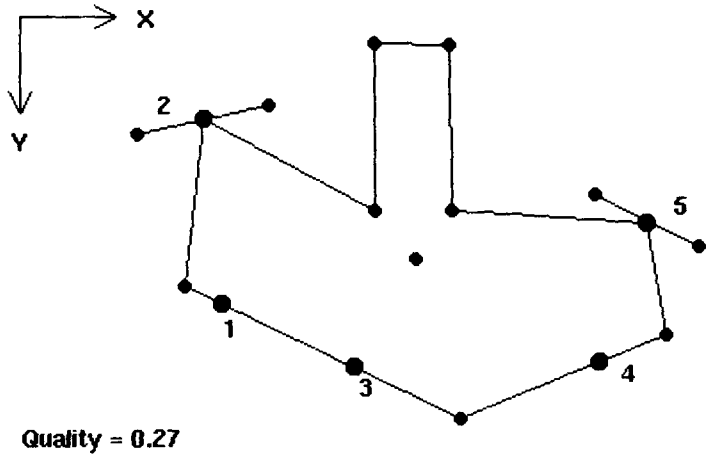


Figure 3.3: A matching grasp of a different object. Contact 2 is no longer between contacts 1 and 3, and the relative positions of all the contact points are very different from those of the example grasp.

outlines the body of this chapter, where a grasp synthesis technique is proposed that involves matching a grasp prototype to a target object. A variety of examples are presented in Chapter 4 to illustrate the properties of this solution technique.

3.1 Problem Statement

This chapter utilizes an abstract and limited definition of a grasp. It presents a solution technique that works within these limits to synthesize a good grasp of a target object by matching a grasp prototype to the object. To fully define the problem solved in this chapter, the following specifications are needed. Modifications from the problem statement of the previous chapter are shown in italics:

Definitions:

- **A grasp:** a set of contacts on a target object.
- **A task:** defined by the task wrench space, or the space of resultant forces and torques that must be applied to the target object.
- **A grasp prototype:** *an example object and a high-quality grasp of that object.*

Inputs:

- A geometric model of a target object,
- *a grasp prototype,*
- a task.

Assumptions:

- Frictionless point contacts,
- no singular contacts,
- no complex contacts,
- contact torques defined about the target object center of mass,
- *grasp quality computed as in Chapter 2.*

Problem:

- *Find a high-quality grasp of the target object.*

The next few paragraphs elaborate upon these specifications. In particular, they describe how grasp quality is computed for a proposed grasp, and they review some assumptions related to the use of a grasp prototype in the process of grasp synthesis. As in Chapter 2, it is important to note that although two-dimensional illustrations are used throughout the chapter, the techniques described are general. Section 3.7 covers the trivial extensions required to accommodate three-dimensional objects.

Grasp Quality can be Computed from the Contacts of a Grasp

As in Chapter 2, the forces and torques that can be exerted at the contacts of a grasp can be completely specified from a description of that grasp and from the assumptions presented above. The direction of applied force can be uniquely determined at a contact, because non-singular frictionless point contacts are assumed. Torque can be determined from the applied force, the contact point, the target object center of mass, and the torque multiplier (see Section 2.2).

Once the forces and torques that can be applied to the target object at the contacts of a grasp are known, a grasp quality measure can be computed for the given task as described in Chapter 2. Throughout most of this chapter, the very general grasp quality measure of Equation 2.25 will be assumed. This measure was designed to approximate the worst case efficiency with which the grasp can counter a disturbance force applied to the target object. Although this is a good measure to use for illustrating the grasp synthesis technique of this chapter, it captures none of the characteristics specific to a particular task. Section 3.6.5 below shows how the techniques demonstrated in this chapter can be used with any given task.

Grasp Prototypes are Used in the Solution

Given a grasp quality measure, the problem of this chapter becomes one of finding a set of contacts with the target object that produces a high grasp quality measure. The sections below show how a grasp prototype can be used to help find high quality grasps. The prototype will constrain the search for contact points on the target object.

One important part of the process of applying a grasp prototype to a new target object is to find an appropriate configuration of the grasp prototype with respect to the target object, as both have their own local coordinate systems. It is assumed in this report that the prototype coordinate frame is fixed to the target object center of mass, although it is free to rotate about this point. This assumption is convenient because good solutions will generally fall around this point. If the grasp prototype is assumed to be fixed to the target object center of mass, there are fewer parameters left to specify. This assumption is not necessary, however.

The problem stated here assumes that a grasp prototype has already been selected. For the purposes of this report, the grasp prototype is provided as an input parameter, and this chapter concentrates only on fitting it to the target object. Selection of a suitable grasp prototype is a very real problem, however, especially if we expect that a sizable library of grasps will be required to provide a rich sampling of the capabilities of a particular robot

Reference	Objective Function	Task	Constraints
[8]	$\min \mu$	force closure	2 contact, 2D, curved object
[4]	$\min \mu$	force closure	2 contact, 2D, curved object
[20]	$\min \mu$	lift object	3 contact, 3D, polyhedron, hand in contact
[31]	$\min \sum_i \lvert \underline{f}_i \rvert$	lift object	3 contact, 2D, polygon
[31]	$\min \sum_i \lvert \underline{f}_i \rvert$	lift, rotate object	4 contact, 2D, convex polygon

Figure 3.4: Grasp quality measures used for grasp synthesis. The objective functions given measure the suitability of a grasp for the specified task. The last column lists constraints on the problem domains required for the solutions proposed in these references.

hand. Throughout the examples of this and the next chapter are problems that have been designed to explore some of the relevant issues, such as how different a target object can be from a prototype before the prototype is no longer useful. A discussion of the topic of prototype selection is found in Section 8.1.4, and results from the relevant examples are reviewed as part of the discussion in that section.

3.2 Previous Work

This chapter addresses the problem of grasp synthesis. Approaches to this problem can be divided into two broad groups: one focused on contact placement and the other focused on hand shape. Because this chapter uses the abstraction of a grasp as a set of contacts with the target object, only the first group of work, focused on optimal placement of these contacts, is relevant here. The second group of work, focused on matching object shape to hand shape, is reviewed in Chapter 5.

Previous work developing grasp quality measures was discussed in Section 2.3. Most of the grasp quality measures described in that section were used in global or local searches for optimal quality grasps. Most of the section focused on grasp quality measures used to generate grasps optimized with respect to the forces and torques that could be applied to the target object and the match of these forces and torques to those required for a given task. The table outlining objective functions, tasks, and constraints on applicability of proposed grasp quality measures is repeated here for reference (Figure 3.4). Fast techniques for optimizing the given objective functions are found within the references given.

Section 2.3 also stated that grasps can be optimized based on robustness of the contact targets, or based on contact region size. In particular, Ponce et al. [41] show how to construct optimal four-contact grasps of polyhedral objects using linear programming, Faverjon and Ponce [13] construct optimal two-contact grasps of curved two-dimensional

objects, and Nguyen [35] describes how to identify a variety of robust contact configurations, including three-contact grasps of polyhedral objects.

In addition to the work based on grasp quality measures, Mishra et al. [32] show that 12-contact force closure grasps of three-dimensional objects can be found in $O(n)$ time, although the grasps that are found are not in general optimal by any measure.

All of these methods for solving the grasp synthesis problem employ harsh constraints on the types of grasps that can be generated. In particular, the solutions may apply only to two-dimensional objects, they may require very small or very large numbers of contacts, and they may limit contact to the edges of a polygon or the faces of a polyhedral object.

If the main goal in finding good contact placements is to construct a stable, whole-hand grasp of an object, then we will need more than just a few contacts, we will not be able to achieve very large numbers of contacts, and we will want to make contact on features other than the faces of the target object. Constraining the search space by restricting grasps to match a prototype can result in more practical limitations than the solutions described above. The sections and chapters below show that the process of matching to a grasp prototype is sufficiently constraining to allow the solution space to be searched and sufficiently flexible to allow a variety of high-quality solutions to be identified.

3.3 Chapter Overview

The remaining sections of this chapter present a novel algorithm for grasp synthesis. The goal is to use a grasp prototype to reduce the search space of the grasp synthesis problem, without sacrificing the quality of the grasps that are constructed. It is desirable to allow as much flexibility as possible given these goals. One form of flexibility is in the description of the grasp prototype. It should be possible to make effective use of any good grasp that is proposed as a prototype, no matter how strange its appearance. A second form of flexibility is flexibility in target object geometry. The range of target objects for which a high quality grasp can be constructed from a given grasp prototype should be as broad as possible without sacrificing search speed or grasp quality.

Section 3.4 provides some background by describing a brute force approach to the problem of grasp synthesis defined in this chapter. An analysis of this approach reveals the complexity of this problem and provides motivation for some means of reducing this complexity. Section 3.5 proposes a naive technique for improving the speed of the brute force grasp synthesis algorithm through use of a grasp prototype. This naive technique does not create a clean division between solutions that do and do not match the grasp prototype, and it actually expands the size of the space within which solutions might be found. An analysis of these problems motivates an additional set of design goals, and Section 3.6 presents a grasp synthesis algorithm that meets these design goals. This discussion is illustrated using a very simple two-dimensional object, and Section 3.7 covers the simple extensions required to move to arbitrary three-dimensional objects. Section 3.8 concludes with an evaluation of the grasp synthesis technique presented in this chapter.

3.4 Examining all Contact Assignments

This section describes a brute force approach to synthesizing a high quality grasp of a target object. This approach does not use a grasp prototype to guide the search, but an analysis provides some background and motivation for the use of a grasp prototype. The paragraphs below first describe the search space of the problem and then show how optimal grasps can be extracted from this space.

The grasp synthesis problem of this chapter can be expressed as the problem of selecting a set of contacts that will form a grasp. For the purpose of this discussion, it is assumed that grasps will be formed that have no more than c contacts.

To place a single contact requires selecting a feature of the target object on which this contact will be placed, and finding an exact placement of the contact on the selected feature. As in Chapter 2 (Figure 2.6), a two-dimensional target object can be described in terms of the following types of features: curved edges, straight edges, and vertices. To place a contact on any one of these features requires a single parameter. On a curved or straight edge, the position of the contact must be specified. If this position is known, then the local normal of the contact can be determined, and a contact wrench can be computed. On a vertex of the target object, the contact position is already known, but the orientation of the contact force must be specified before the contact wrench can be computed.

The complete space of grasps that are possible for a given target object can be described as follows. For each of the c contacts, a contact feature must be chosen and a single parameter provided to fully describe the position and local normal of the contact. Contact ordering does not matter, however. All grasps having contacts in the same locations will have the same grasp quality measure. If the target object boundary curve (Figure 2.12) is quantized into p sections, the size of the search space can be described as:

$$\begin{pmatrix} p \\ c \end{pmatrix} = \frac{p!}{c!(p-c)!}. \tag{3.1}$$

From Chapter 2 we know that each of these grasps can be associated with a grasp quality measure. To compute a grasp quality measure over this space as specified in Equation 2.25 would require constructing and testing a convex hull for each possible grasp.

Optimal grasps can be extracted from the search space by considering:

- the grasp quality measure, and
- variation in this grasp quality measure with errors in contact placement.

The grasp with the optimal grasp quality measure is theoretically the best grasp. This grasp tends to be fairly robust, but in some cases small errors in the target object model or small errors in placing the contacts can drastically reduce the quality of the grasp. The effect of small errors in contact placement can be determined by examining how these errors affect the grasp quality measure. Grasp robustness can be improved by finding large

connected regions of the search space having acceptable quality measures, and choosing a solution well within one of these regions.

Although it may be possible to develop more efficient search solutions, a naive approach to finding an optimal grasp using grasp quality and grasp robustness measures would in general require examining much of the search space. This will become difficult if there are more than two or three contacts. This is the reason for the very aggressive restrictions on both the problem domain and the solution space reflected in previous work. In some special cases, clever techniques can be found to get around the combinatorics of the general problem.

3.5 Prototypes can Reduce Quality Computation

One thing that makes the brute force search for a high quality grasp expensive, aside from the extremely large size of the search space, is the fact that a grasp quality measure must be computed at each point in this search space. Computing grasp quality is an expensive operation, as it requires computing a convex hull. This section shows that the use of a grasp prototype can make this computation unnecessary. In particular, a class of grasps representing variations on the grasp prototype can be defined such that:

- *all* grasps within this class have acceptable grasp quality measures, and
- it is much more efficient to determine whether a grasp is a member of this class than to compute the grasp quality measure from scratch.

Although the technique described in this section is not practical, it serves to introduce the grasp prototype and to motivate the approach described in the next section, which forms the basis of this chapter.

The intuition explored in this section runs as follows. The grasp prototype, by definition, is an example of a successful grasp. If the target object is similar to the prototype object, then it should be possible to find a high-quality grasp of the target object that resembles the grasp prototype. If an appropriate grasp prototype can be easily selected, this heuristic has many advantages. For example, arbitrary constraints such as those limiting allowable types of contacts or allowable numbers of contacts are avoided in the sense that any given grasp prototype can be used as a reference point for the search.

Effective use of a grasp prototype requires a compact definition of the class of grasps matching that prototype. It might seem desirable to define classes of grasps based on their appearance. A continuous range of contact positions can, for example, be associated directly with a continuous family of target object geometries. This approach is reviewed in Section 5.3. It is appealing because grasp synthesis requires only that the target object family be recognized. An appropriate grasp can then be extracted from the class of grasps associated with that target object family. Unfortunately, this approach has only been demonstrated for very simple families of target objects, and the problem of adapting the resulting grasps to accommodate unexpected variations in target object geometry or to respond to problems caused by obstacles has not been adequately addressed. In addition,

a target object that is not a member of any recognized family cannot be grasped at all using this approach.

This report explores a different approach to using a grasp prototype. This approach does not rely on an a priori assignment of grasps to target object geometries. Instead it classifies grasps based on the distribution of contact wrenches that can be generated at the contact points that make up a grasp. This is a convenient way to classify grasps because the classification is related directly to the grasp quality measure. The sections below show that this approach creates a membership function that is easy to evaluate, guarantees grasp quality in all situations, and can be applied to any target object geometry.

One very simple way to classify grasps based on contact wrench distribution is to define an error term that limits the acceptable distance of the contact wrenches of a proposed grasp from the contact wrenches of the prototype. Computing this error term is much more efficient than computing the grasp quality measure. To preserve grasp quality, the allowable error is limited so that all grasps with an error term below a certain error threshold have a grasp quality above a given quality threshold. The equation below illustrates an example error term $E(\underline{w})$. In this equation, c is the number of contacts of the prototype, $\underline{w}$ is the vector of individual contact wrenches $\underline{w}_i$ of a proposed grasp, and $\underline{W}_i$ represents contact wrench i of the grasp prototype.

$$E(\underline{w}) = \sum_{i=1}^{c} |\underline{w}_i - \underline{W}_i|^2. \tag{3.2}$$

For a given task, any set of contact wrenches $\underline{w}$ can be associated with a grasp quality measure, represented by function $Q(\underline{w})$. To limit error term $E(\underline{w})$, a grasp quality threshold q' is chosen that is less than or equal to the quality measure of the grasp prototype q, and a value $\epsilon(q')$ is found such that:

$$(E(\underline{w}) \leq \epsilon(q')) \Longrightarrow (Q(\underline{w}) \geq q'), \quad q' \leq q. \tag{3.3}$$

Figure 3.5 illustrates this expression. The set of wrenches representing the grasp prototype is shown with a dot in the figure. Any wrench space vector within the error ball drawn around the grasp prototype has a quality measure greater than or equal to q'.

Expression 3.3 is always satisfied at $\epsilon(q') = 0$. Error term $E(\underline{w})$ cannot be less than 0, so if it is equal to 0, then the contact wrenches of the grasp are identical to those of the grasp prototype ($\underline{w}_i = \underline{W}_i$). This means that $Q(\underline{w})$ will be the quality q of the grasp prototype, which is greater than or equal to q' by definition. If q' is set to be strictly less than q, then unless the grasp is at some sort of singularity, we expect to find a value of ϵ greater than 0 that satisfies Expression 3.3. Then the set of grasps that satisfy the expression $(E \leq \epsilon(q'))$ can be said to match the grasp prototype for a grasp quality threshold of q'. The grasp quality measure is being used to define equivalence classes of grasps about a single grasp prototype.

Expression 3.3 can be used to synthesize high quality grasps of the target object by selecting a grasp quality threshold q' and finding the corresponding error threshold $\epsilon(q')$. Then error term $E(\underline{w})$ can be computed for all possible grasps. Any grasp having an error

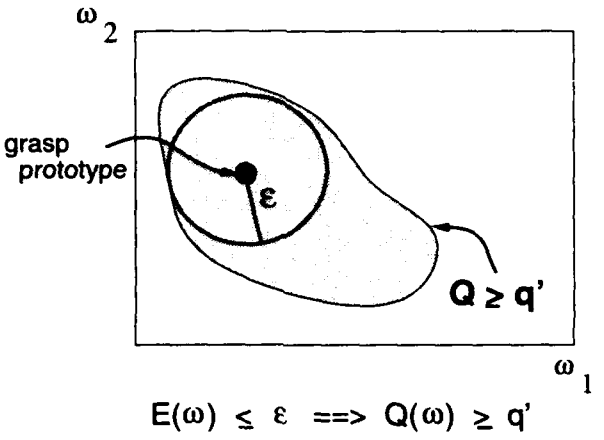


Figure 3.5: A simple technique for defining an equivalence class of grasps about a single example grasp, or grasp prototype. This equivalence class of grasps is defined so that all grasps within the class have an acceptable grasp quality measure. The technique involves surrounding the example grasp with an error ball in wrench space.

term less than or equal to $\epsilon(q')$ will have a grasp quality measure greater than q'. Because computing error term $E(\underline{w})$ is more efficient than computing the grasp quality measure for a grasp, the grasp prototype has allowed for a more rapid evaluation of any given grasp.

Unfortunately, this use of a prototype may not result in a search algorithm that is faster than the brute force algorithm of the previous section. Although computing the error term $E(\underline{w})$ is faster than computing a grasp quality measure, the search space of possible grasps has become larger.

First, before the error term $E(\underline{w})$ can be computed, an assignment of contacts on the target object to contacts on the example grasp must be specified. This ordering of contacts was not required in the original problem, since the grasp quality measurement is independent of contact ordering, and so a factor of $c!$ more grasps are present in the search space of this new problem.

Second, before the error term $E(\underline{w})$ can be computed, a transform must be specified between the coordinate systems of the target object and the grasp prototype. The value of this transform affects the relative orientation of the applied forces of the proposed grasp and the applied forces of the grasp prototype, and so $E(\underline{w})$ will certainly vary with the value of this transform. The relative *position* of the target object and grasp prototype has been fixed, because it was assumed in the problem specification of Section 3.1 that a grasp is always defined about the center of mass of the target object. The relative *orientation* of the target object and grasp prototype is not known, however. This orientation term is required to compute the error term proposed above, but it is not required to compute grasp quality, so the orientation dimension must be added to the search space of the problem.

Given the increased size of the search space, this method of using a grasp prototype to focus the search for a high-quality grasp of a target object does not appear to be exceptionally useful. While a mechanism has been provided for efficiently determining that a grasp has an acceptable grasp quality measure, the search space of the problem has been expanded by the addition of parameters required to specify contact ordering and grasp prototype alignment. Although heuristics could be developed to indicate where it might be promising to compute the error term, this expansion of the problem search space is a matter of concern. The next section describes a different technique for defining equivalence classes of grasps about a grasp prototype. This technique is shown to be very effective in reducing the space that must be searched to find an optimal grasp matching the prototype.

3.6 Prototypes can Reduce Search Complexity

The previous section presented a method for using a grasp prototype to determine whether a proposed grasp of a target object was guaranteed to meet a given grasp quality threshold without computing the grasp quality measure for the grasp. Unfortunately, the additional parameters introduced served to expand the problem search space, negating some of the benefits of using the prototype. In this section, the use of a grasp prototype is modified to reduce the size of the search space despite the additional parameters needed to align the grasp prototype to the target object.

3.6.1 Introduction

In the sections above, the size of the grasp synthesis problem led to the proposal of constraining this problem by looking only at grasps that match a grasp prototype. The class of grasps matching a grasp prototype should be as general as possible while preserving grasp quality and allowing rapid identification of grasps within this class.

The previous section used a simple error function to match a grasp to a prototype. This approach was not effective in reducing the combinatorics of a search through the space of possible grasps. The algorithm of this section proposes to overcome this problem by defining sets of constraints to be applied to the contacts of the grasp independently. The set of grasps matching a prototype is described using these independent *contact constraint sets* such that if each set of contact constraints is satisfied by some contact wrench of a grasp, then the grasp will have a grasp quality measure above a certain threshold. When each contact of the grasp can be independently evaluated, the combinatorics of the grasp synthesis problem are greatly reduced.

In general it would not be possible to evaluate contact wrenches independently, because all contact wrenches contribute to the quality measure of a grasp. By looking at a single contact we cannot evaluate grasp quality or even guess whether a grasp will be stable. The sections below show that when contact constraint sets are constructed around a grasp prototype, these estimates can be made.

To see how the grasp prototype might help in forming independent contact constraint sets, imagine the following experiment. Suppose that a grasp prototype is given that is a c-contact grasp with quality measure q. Imagine leaving the first $c-1$ of the contacts fixed and asking what values of contact wrench c would result in a grasp with a grasp quality measure of at least q', where $q' \geq q$. We know one contact wrench that will work: contact wrench c of the grasp prototype. In the general case, however, there will be a range of other contact wrench values that work sufficiently well. By describing this range of contact wrench values, we form a contact constraint set for contact c and grasp quality threshold q'.

The object of this section is to show how independent contact constraint sets can be formed for all contacts of the grasp simultaneously. The union of contact constraint sets for a given grasp prototype and grasp quality threshold represents a class of grasps having quality measures greater than the given threshold. This class of grasps is called a *generalized grasp prototype.*

The generalized grasp prototype would be even more useful if a grasp could be optimized by optimizing the contacts of that grasp independently. In other words, in place of a contact constraint set based on a given grasp quality threshold, a *contact quality measure* could be defined based on the family of contact constraint sets defined over the entire space of grasp quality thresholds. If contact quality measures can be optimized independently, it is not necessary to worry about selecting an appropriate grasp quality threshold for the contact constraint sets.

There are three new terms that will be described in detail in this section. For reference, definitions of these terms are shown below:

- **Generalized grasp prototype:** a class of grasps, defined around a grasp prototype, such that every grasp within this class has a quality measure exceeding a given threshold. A generalized grasp prototype is composed of contact constraint sets, one for each contact of the grasp.
- **Contact constraint set:** a set of constraints that a contact wrench must satisfy to be considered a component of a successful grasp of a target object. The contact constraint set is the building block of a generalized grasp prototype. For a grasp to match the description of a generalized grasp prototype, each contact constraint set of that generalized grasp prototype must be satisfied by some contact wrench of the grasp.
- **Contact quality measure:** a measure of the greatest grasp quality threshold for which a given contact wrench would fall within a given contact constraint set of a generalized grasp prototype.

3.6.2 Section Overview

The paragraphs below show how the generalized grasp prototype, contact constraint sets, and contact quality measures can be constructed. Section 3.6.3 begins by describing a generalized grasp prototype that will work for a task wrench space of any shape. Section 3.6.4

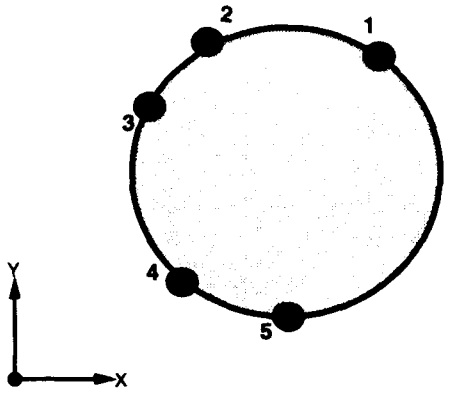


Figure 3.6: An example circle grasp, represented by applied forces at the numbered contact points.

shows how this generalized grasp prototype can be tailored to the simple task wrench space of Equation 2.25. Section 3.6.5 shows that a similar procedure can be followed for any given task wrench space, and Section 3.6.6 considers the problem of generating contact quality measures. Section 3.6.7 shows how the generalized grasp prototype can be used to synthesize optimal grasps of a target object, and Section 3.6.8 presents a summary.

For all of these sections, constructions are presented using a very simple example, the grasp of a circle shown in Figure 3.6. This is a good example because a circle grasp has only a two-dimensional wrench space (Figure 3.7); no torques can be generated in a frictionless grasp of a circle. A two-dimensional wrench space is used because it can easily be visualized, but all of the expressions and equations that are presented are general and apply to wrench spaces of any dimension as described in Section 3.7.

3.6.3 Contact Constraint Sets Independent of Task

This section shows how to construct a generalized grasp prototype that preserves the grasp quality of a prototype for any task wrench space. That is, even if the task wrench space is not known when the generalized grasp prototype is constructed, the generalized grasp prototype is still guaranteed to represent a class of grasps that are at least as good as the original prototype.

This section begins by reviewing the definition of grasp quality. The grasp quality measure is defined as the scale of the task wrench space required for this task wrench space to just fit within the unit grasp wrench space. To construct a new grasp with a grasp quality measure at least as high as that of a given grasp prototype, it is necessary to ensure that the unit grasp wrench space of the new grasp contains a task wrench space of the same scale as that contained by the unit grasp wrench space of the grasp prototype.

If a task wrench space is not available, then in order to ensure that the grasp quality

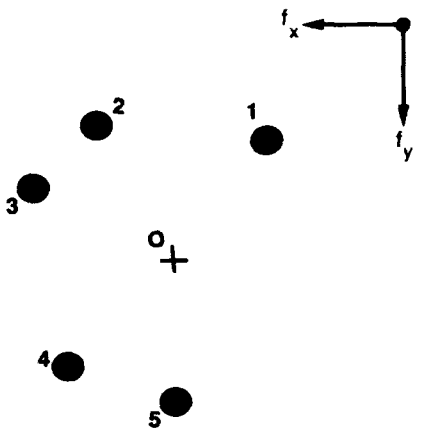


Figure 3.7: A circle grasp has a two-dimensional wrench space. No torques can be applied to a circle in a frictionless grasp of that circle. This figure shows the x and y components of the unit forces that can be applied to the circle from the contact points shown in Figure 3.6.

of the new grasp will be as good as that of the grasp prototype for *any* given task wrench space, it is necessary to ensure that the grasp wrench space of the new grasp contains the entire grasp wrench space of the prototype. Recall that the grasp wrench space is given by the convex hull of the contact wrenches of a grasp (Section 2.5). This means that the convex hull of the contact wrenches of the new grasp must contain the convex hull of the contact wrenches of the prototype. To construct a generalized grasp prototype, then, we can construct a wrench space region about each contact wrench of the grasp prototype such that the set of wrench space regions has the following properties:

- Each contact of the grasp prototype corresponds to and is contained within a wrench space region.
- If a grasp is constructed with one contact wrench in each region, then the convex hull of the contact wrenches of this grasp will contain the convex hull of the contact wrenches of the grasp prototype.

This will certainly satisfy the conditions posed for this problem. If the convex hull formed from the contact wrenches of any new grasp contains the convex hull formed from the contact wrenches of the grasp prototype, then the grasp wrench space of that new grasp will contain the grasp wrench space of the grasp prototype, and no matter what task wrench space is used, the grasp quality measure of the new grasp will be at least as high as that of the grasp prototype.

For the example grasp used in this section, the contact wrenches of the grasp prototype are shown in Figure 3.7, and Figure 3.8 shows a set of wrench space regions that satisfy the

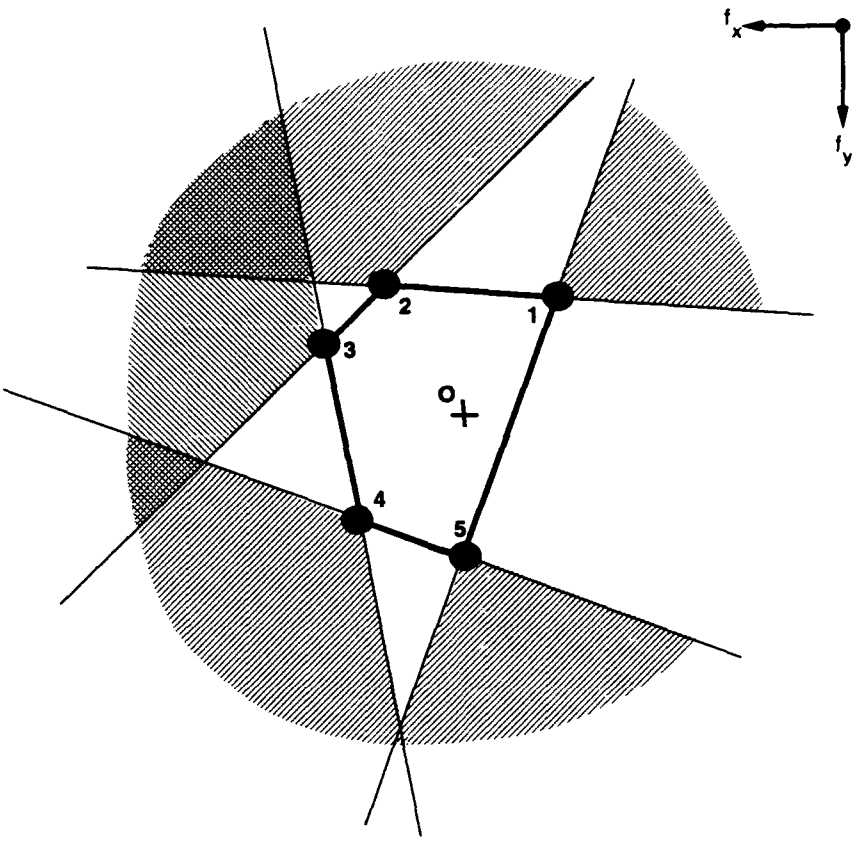


Figure 3.8: Independent wrench space regions that preserve the grasp wrench space of the grasp prototype. Any combination of one wrench in each region will result in a convex hull that contains the complete grasp wrench space of the grasp prototype.

conditions stated above. The regions are bounded by lines containing the segments of the convex hull of the contact wrenches of the grasp prototype. Any new grasp that is formed must have a wrench space point corresponding to each of the original contact wrenches, and each of these points must lie within all of the outer half-space regions defined by the facets of the convex hull passing through the corresponding original contact wrench.

To make this explicit, we define the following parameters from the convex hull of the grasp prototype:

$$\begin{aligned} \underline{w}_i &= \text{contact wrench } i \text{ of the grasp prototype,} \\ g_j &= \text{facet } j \text{ of the grasp prototype convex hull,} \\ \underline{\hat{n}}_j &= \text{unit outward pointing normal of facet } g_j. \end{aligned}$$

Let G_i represent the set of facets containing $\underline{w}_i$ as follows:

$$G_i = \{j : g_j \text{ contains } \underline{w}_i\}.$$

Each wrench of the new grasp $\underline{w}'_i$ is placed so that it meets the following conditions:

$$\forall j \in G_i, \quad \left[(\underline{w}'_i \cdot \underline{\hat{n}}_j) \geq (\underline{w}_i \cdot \underline{\hat{n}}_j)\right]. \tag{3.4}$$

Equation 3.4 represents the contact constraint set for contact i. To show that these constraints are sufficient, imagine any grasp constructed by selecting a contact wrench within each of the wrench space regions shown in Figure 3.8. The convex hull formed from these contact wrenches may not have the same ordering of facets or even the same number of facets as the original convex hull (e.g. see Figure 3.9), but it does contain the original convex hull. A sketch of a proof follows.

1. *No facet of the new convex hull intersects the interior of the original convex hull.* Suppose that there is some such facet. Construct the hyperplane containing this facet and call it P. Hyperplane P divides space into inner and outer halfspaces, and by definition of a convex hull, all points of the new convex hull will either lie within hyperplane P or in the inner halfspace of P. Since P intersects the interior of the original convex hull, by definition of a convex hull, there must be at least one vertex of the original convex hull in the outer halfspace of P. By definition of the contact constraint sets in Equation 3.4, a vertex of the original convex hull that is perpendicularly furthest to the outside of P will have its entire contact wrench region contained in the outer halfspace of P. By construction of the new grasp, there is a contact wrench within this region. This contact wrench is in the outer halfspace of P. $\Rightarrow\Leftarrow$

2. *The new convex hull contains some point in the interior of the original convex hull.* Choose any point in the interior of the original convex hull and call it p. Suppose the the new convex hull does not contain p. Then there exists some hyperplane P that divides space into inner and outer halfspaces and passes through p, such that all wrenches of the new convex hull either lie within hyperplane P or in the inner halfspace of P. The grasp prototype by definition *does* contain p. P must

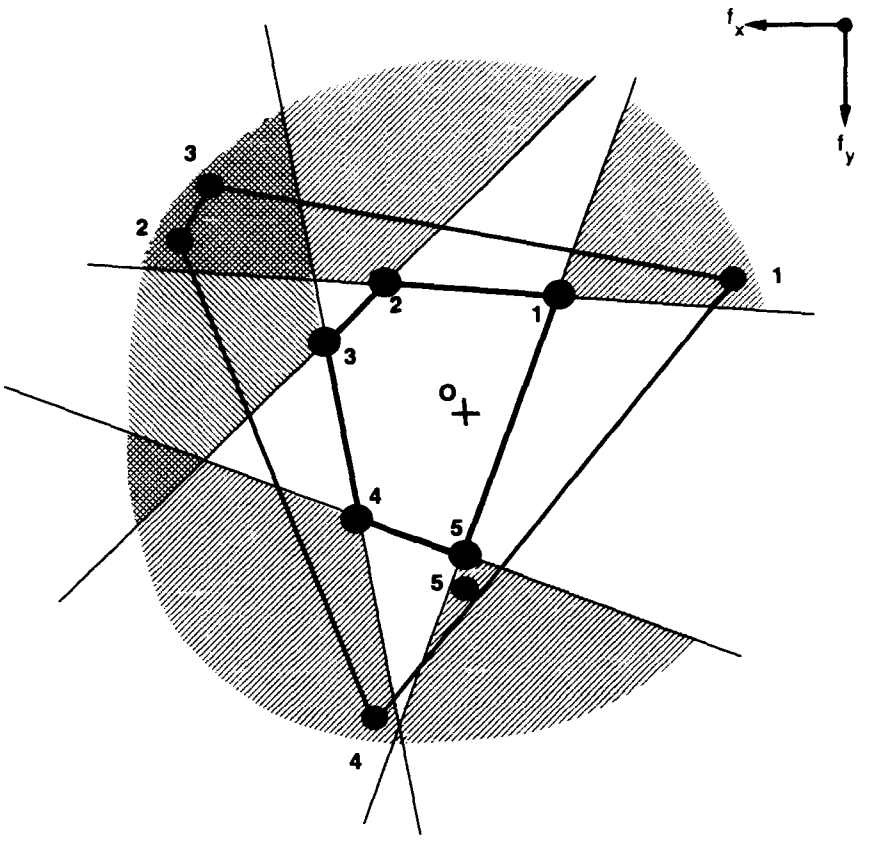


Figure 3.9: Any collection of wrenches within the wrench space regions given will produce a convex hull that contains the complete grasp wrench space of the grasp prototype.

then intersect the interior of the original convex hull. By definition of a convex hull, the outer halfspace of P contains some vertex of the original convex hull. By the argument in 1 above, there is at least one point of the new convex hull within the outer halfspace of P. $\Rightarrow\Leftarrow$

3. *The new convex hull contains the original convex hull.* By 2 above, the new convex hull contains some part of the interior of the original convex hull. If no facet of the new convex hull can intersect the interior of the original convex hull, then the new convex hull must contain the entire interior of the original convex hull.

To see how well the grasp generalization technique works for the example grasp, we can check what contacts on the circle object match the contact constraint sets given. Unfortunately, only the original contact points satisfy these constraint sets. No other combination of five contacts on the circle object results in a grasp wrench space that contains the entire grasp wrench space of the grasp illustrated by the convex hull in

Figure 3.8. This is true because of the following assumptions stated in Sections 2.2 and 2.5. The grasp wrench space is described in those sections as:

$$\sum_{i=1}^{c} \alpha_i \underline{w}_i, \tag{3.5}$$

subject to

$$\alpha_i \geq 0, \tag{3.6}$$

$$\sum_{i=1}^{c} \alpha_i \leq 1, \tag{3.7}$$

$$|\underline{f}_i| = 1. \tag{3.8}$$

In the two-dimensional wrench space of this example, $\underline{w}_i = \underline{f}_i$, and so for each contact i, $|\underline{w}_i| = 1$. Figure 3.8 shows that within the wrench space regions illustrated, this equation holds only for the original contact wrenches.

To obtain more flexibility in this grasp, the grasp quality requirements will have to be lowered to some fraction of the grasp quality of the prototype. If the prototype has a quality of q, Equation 3.4 can be modified to reflect a grasp quality of q' as follows:

$$\forall j \in G_i, \quad \left[(\underline{w}_i' \cdot \hat{\underline{n}}_j) \geq \frac{q'}{q} (\underline{w}_i \cdot \hat{\underline{n}}_j) \right]. \tag{3.9}$$

The facets in G_i are moved inward to a fraction of $\frac{q'}{q}$ of their original distance from the wrench space origin, and new regions are formed from this new set of facets.

Figure 3.10 shows the set of independent contact regions that results for a grasp quality measure of at least 75% of that of the grasp prototype. As long as this object is grasped *with one contact in each of the regions shown in the figure*, a grasp can be formed having a grasp quality measure of at least $0.75q$ for *any* task wrench space. Note that the contact region sizes in Figure 3.10 are very unequal. Section 3.6.8 explores the possibility of adjusting the sizes of these regions.

3.6.4 Contact Constraint Sets for a Simple Task

The previous section described a very conservative method for generalizing a grasp prototype. It was necessary to be very conservative because it was assumed that there was no knowledge of the actual task wrench space. The previous approach had to form a generalization that was sure to include any task wrench space that the prototype originally included.

This section takes a different approach. Here, knowledge of the task wrench space is assumed, and the task of countering disturbance forces is used as an example. Recall that this results in a ball-shaped task wrench space, as discussed in Section 2.8. The next paragraphs show that a much greater degree of grasp generalization is possible given this task wrench space information.

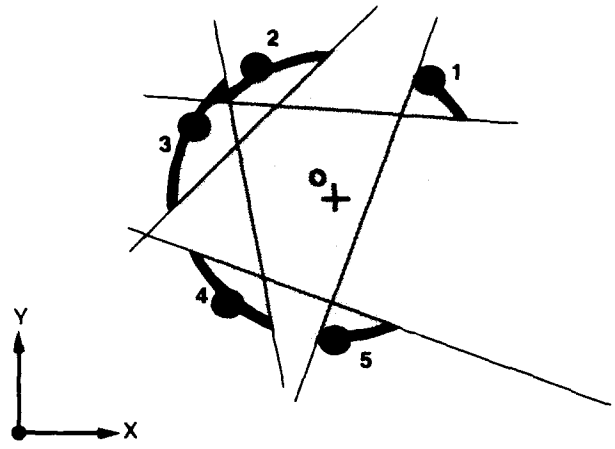


Figure 3.10: Independent contact regions on the circle object such that one contact within each region guarantees a grasp quality measure of 75% of the original grasp quality measure.

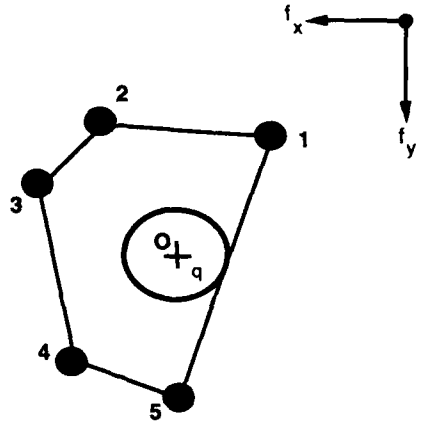


Figure 3.11: A wrench space ball is a very simple task wrench space. The grasp quality measure for this task wrench space is not very high for this grasp.

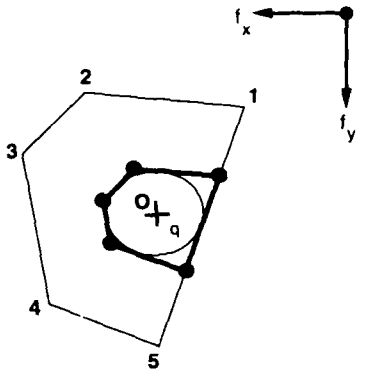


Figure 3.12: An intermediate grasp wrench space can be formed by projecting the convex hull that bounds the complete grasp wrench space onto the task wrench space. Only this intermediate grasp wrench space need be preserved to guarantee the same grasp quality measure as the original grasp.

Figure 3.11 shows the quality measure of the example prototype for this task. The contact constraint sets of the generalized grasp prototype for this quality measure must be defined so that the convex hull formed from the new wrenches contains the grasp quality circle shown in the figure. Much larger wrench space regions can be formed that satisfy this condition. The same method is used as before, but first an intermediate convex hull is generated. To constrain this intermediate convex hull, the new contact constraint sets formed are defined to contain the more general contact constraint sets of the previous section, and so this intermediate hull is constructed from segments tangent to the grasp quality circle and parallel to the segments of the original convex hull (Figure 3.12). The idea that this intermediate hull could be constructed differently is explored in Section 3.6.8. Regions resulting from this intermediate hull are shown in Figure 3.13. The convex hull formed from any set of contact wrenches within these regions will contain the intermediate hull, and it will also contain the grasp quality ball. Compare these to the regions of Figure 3.8.

The constraint equations can be written as follows:

$$\forall j \in G_i, \quad \left[(\underline{w}_i' \cdot \hat{\underline{n}}_j) \geq q \right]. \tag{3.10}$$

Every segment of the original hull is represented on the intermediate hull. Each half-space constraint is dropped in a perpendicular direction toward the origin, not pulled away from it, which means that the regions obtained from the intermediate hull contain the original regions. Larger independent regions are created, but the grasp quality measure is preserved.

To compare this generalized grasp prototype directly to that of the previous section,

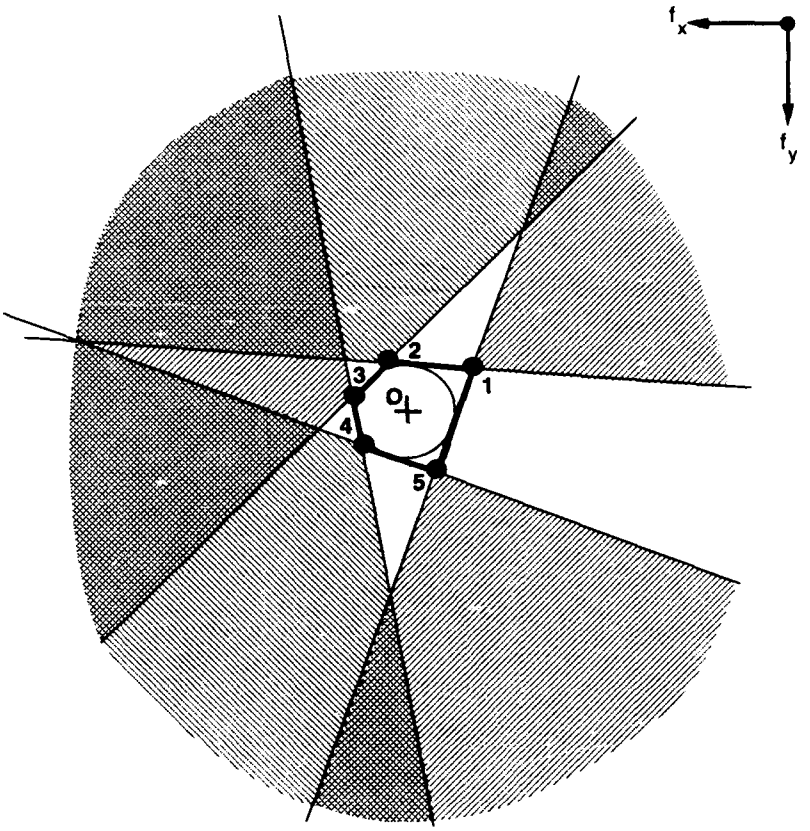


Figure 3.13: Independent wrench space regions that preserve the grasp quality measure of the original grasp. These regions are larger that the wrench space regions of Figure 3.8, because only the wrench space ball must be preserved, not the entire wrench space of the original grasp.

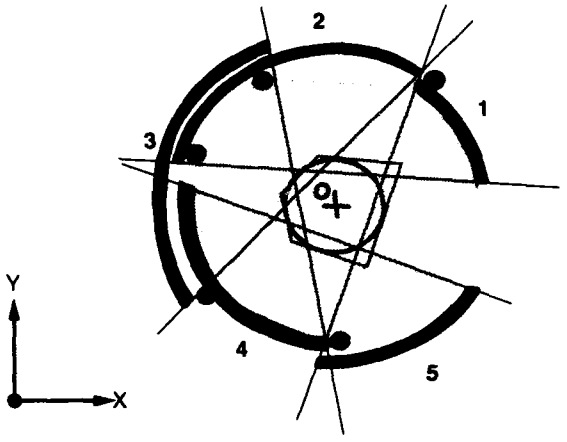


Figure 3.14: Independent contact regions on the circle object such that one contact within each region guarantees a grasp quality measure of 75% of the original grasp quality measure.

Equation 3.10 is rewritten for a reduced grasp quality measure q':

$$\forall j \in G_i, \quad \left[(\underline{w}_i' \cdot \hat{\underline{n}}_j) \geq q'\right]. \tag{3.11}$$

Figure 3.14 shows these new contact constraint sets applied to the example target object for a grasp quality measure of at least $0.75q$. Compare this to Figure 3.10 of the previous section. The regions are much larger with the new contact constraint sets. This illustrates the power of knowing how a grasp will be evaluated.

In some sense, however, this comparison is not fair, as the example grasp is not a very high quality grasp for the simple task of countering disturbance forces. If this were an example of an actual grasp prototype, the grasp would have been designed with some particular task in mind, and it would be expected to be well suited for that task.

3.6.5 Contact Constraint Sets for any Given Task

The previous section described a method for generalizing a grasp based on a ball-shaped wrench space. The obvious shortcoming of that generalization is that it is not applicable to task wrench spaces that differ significantly from a ball. This section describes an extension to the previous algorithm that allows generalization of grasp prototypes for a task wrench space of any shape. This extension is not used in any examples of this report, but the quality measure developed here would be required if a specific task wrench space that was not a wrench space ball were provided as an input to the grasp synthesis problem.

The extension required to the algorithm of the previous section is very simple: every facet of the prototype convex hull is associated with a scale factor that indicates the distance of that facet from the origin when the entire unit task wrench space is just on

the inside of the hyperplane containing that facet. A revised contact constraint set is then constructed as follows. Let the scale factor be γ_j for facet g_j. Then a grasp quality measure of q implies that:

$$\forall j \in G_i, \quad \left[(\underline{w}_i' \cdot \hat{\underline{n}}_j) \geq \gamma_j \, q\right]. \tag{3.12}$$

This technique can be used to improve the grasp quality approximation for the task of countering disturbance forces illustrated in the previous section. Chapter 2 showed how the wrench space ball was a rough approximation of a more intuitive task wrench space: the object wrench space of the target object (see Figure 2.13 of Section 2.6). If the object wrench space of the target object is known, a generalized grasp prototype can be constructed using that object wrench space as the task wrench space.

Unfortunately, a task wrench space based on the object wrench space of a target object has a major shortcoming. This task wrench space varies with target object geometry. The generalized grasp prototype would have to be recomputed for each new target object. It would also have to be recomputed for each new alignment of the grasp prototype to that target object, because the γ_j factors of Equation 3.12 would change with this alignment parameter. Clearly, the problem is that the task wrench space for this task is fixed with respect to the target object, and has nothing to do with the grasp prototype. This is true because the task wrench space reflects properties of the geometry of that target object and not properties associated with the prototype grasp.

In general, however, we might expect to find task wrench spaces that are more fixed with respect to a grasp prototype, and less dependent on target object geometry. This assumption is important if we expect to be able to design grasps to work well for specific tasks, such as the task of drinking from a glass. If the task wrench space can be fixed with respect to the grasp prototype, then a complex task wrench space can be handled much more easily. The generalized grasp prototype can be computed offline and stored as part of a grasp library.

3.6.6 Contact Quality Measures

The previous sections described how to create contact constraint sets by generalizing prototype grasps. Methods for creating contact constraint sets when a task wrench space is known and when it is not known were described. This section shows how to use these contact constraint sets to compute a contact quality measure.

Recall that the contact quality measure is defined as a measure of the greatest grasp quality threshold for which a given contact wrench would fall within a given contact constraint set of a generalized grasp prototype. For any given grasp quality threshold, contact constraint sets were bounded by projections of the grasp prototype wrench space convex hull onto the task wrench space scaled according to that grasp quality threshold (Figure 3.13). This projection of the grasp prototype convex hull onto the task wrench space was called an *intermediate convex hull* for that grasp quality threshold.

This description can be inverted to say that any contact wrench proposed as a match to some contact of the grasp prototype supports a family of intermediate convex hulls over a range of grasp quality thresholds. This idea is illustrated in Figure 3.15 for the

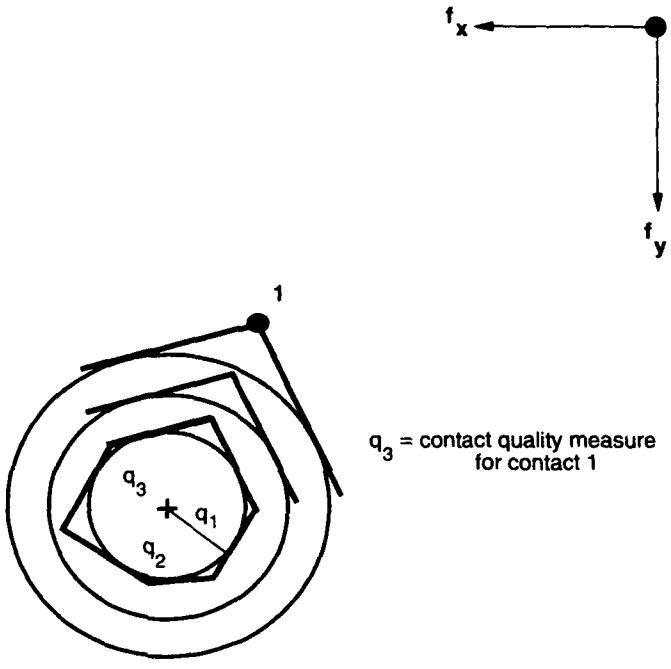


Figure 3.15: Each contact wrench can independently be determined to support its corner of a family of convex hulls corresponding to a family of grasp quality measures.

simple ball-shaped task wrench space. The contact quality measure of any contact is the maximum grasp quality threshold for which an intermediate convex hull can be supported by that contact.

To optimize the contact quality measure of each contact for a task wrench space that is a wrench space ball, simply place each new contact wrench $\underline{w}_i'$ such that

$$\min_{j \in G_i} (\underline{w}_i' \cdot \hat{\underline{n}}_j) \tag{3.13}$$

is maximized.

A lower bound on the quality of the grasp can then be stated as:

$$Q_{min} = \min_i \left[\min_{j \in G_i} (\underline{w}_i' \cdot \hat{\underline{n}}_j) \right]. \tag{3.14}$$

An intermediate convex hull of scale Q_{min} can be supported by all contacts.

To maximize the contact quality measure for a more complex task wrench space, a weighting factor γ_j is needed for each facet g_j of the intermediate convex hull. Each new contact wrench $\underline{w}_i'$ should be placed to maximize

$$\min_{j \in G_i} \left[\frac{(\underline{w}_i' \cdot \hat{\underline{n}}_j)}{\gamma_j} \right], \tag{3.15}$$

and a lower bound on grasp quality for this task shape is

$$\min_i \left(\min_{j \in G_i} \left[\frac{(\underline{w}_i' \cdot \hat{\underline{n}}_j)}{\gamma_j} \right] \right). \tag{3.16}$$

The technique just described optimizes the quality of a grasp by independently optimizing the quality of each contact wrench. This is a very powerful approach. The ability to separate and localize the computation of contact quality measures is a key feature of the algorithms described in Chapters 5, 6, and 7.

3.6.7 Optimizing Contact Placement

The contact constraint sets and contact quality measures described above can be used to form optimal grasps in two ways, emphasizing slightly different properties of a grasp. These properties are:

- robustness to errors in contact placement for a given grasp quality threshold, and
- a lower bound estimate of the grasp quality measure.

Synthesis of optimal grasps based on either of these properties requires a representation of contact quality space. The paragraphs below first describe how this space can be computed and then outline grasp optimization techniques based on the two properties listed above. These techniques are demonstrated in Chapter 4.

Computing Contact Quality Space

It is useful to sketch the dimensions of the contact quality space. A contact quality measure can be computed as described above for every contact of a grasp, for every position of that contact on the target object boundary curve, and for every orientation of the grasp prototype with respect to the target object.

Figure 3.16 shows a plot of a thresholded contact quality space for contact 1 of the five-contact grasp in the top half of the figure, given a task wrench space that is a wrench space ball. The object boundary curve of the polygon was sampled at regular intervals, creating a set of 609 sample points for every orientation of the grasp prototype with respect to this polygon. Each sample along the object boundary curve corresponds to a unique contact wrench. Combinations of prototype orientation and contact wrench that result in a contact quality measure greater than zero are highlighted in the plot. The tails on the highlighted regions are an artifact of the fact that a scale factor must be chosen to relate the change in angle at a vertex contact to the change in position at an edge contact. In this plot, that factor caused edge length to be sampled finely in comparison with vertex angle ranges.

For a c contact grasp and a target object with an object boundary curve sampled p ways, computation of the entire contact quality space requires cp contact quality computations for each sampled orientation in S^1, where S^1 is the unit circle.

Finding Robust Grasps

Contact quality spaces such as that shown in Figure 3.16 can be used to find robust grasps of a target object. Suppose that an acceptable quality threshold has been chosen. Then the size of the largest continuous region of contact points exceeding the given grasp quality threshold can be found at any single orientation of the grasp prototype with respect to that target object (i.e. at any single point on the x-axis of Figure 3.16). This number is related to the allowable error in placement of the given contact. Figure 3.16 represents a threshold of 0 quality measure. Given this threshold, there are contact regions of 66 units at an orientation of π and 23 units at an orientation of $\frac{3\pi}{2}$. The first of these represents a substantial fraction of the object boundary curve, and so a prototype orientation of π corresponds to a large contact region for contact 1.

The robustness measure for a given contact can be written $R(i,\theta)$, a function of the contact (i) and the relative orientation of the grasp prototype and target object (θ). To find an optimal grasp using this measure requires minimization over the contacts of a grasp and maximization over the orientation parameter:

$$R_{max} = \max_{\theta} \left(\min_{i} R(i,\theta) \right). \tag{3.17}$$

The best grasp will be found at the value of θ producing R_{max}. This may be different for different grasp quality thresholds.

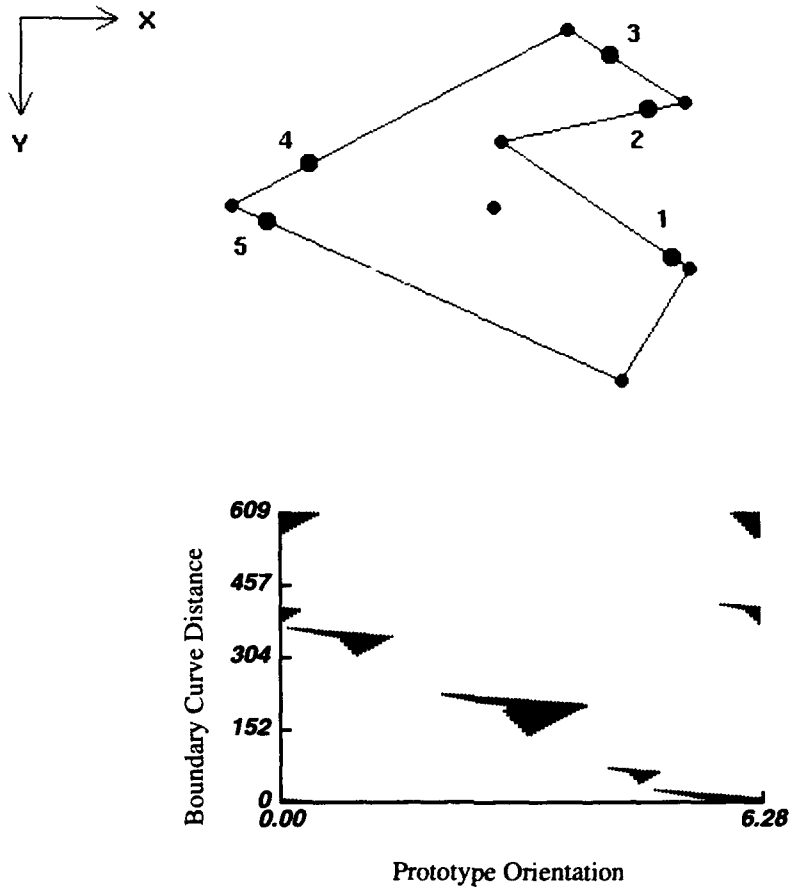


Figure 3.16: The bottom figure shows a plot of the space of positive contact quality measures for contact 1 of the grasp shown in the top figure. The region or regions of good contact points can be large, small, or nonexistent, depending on the orientation of the grasp prototype with respect to the target object. A good grasp requires an orientation that has reasonably sized contact regions for all five contacts.

Finding Optimal Quality Grasps

Contact quality spaces such as that shown in Figure 3.16 can also be used to find optimal quality grasps. Contact quality can be expressed as $Q(i, \theta, s)$, a function of contact (i), relative orientation of grasp prototype and target object (θ), and position on the object boundary curve (s). Then the optimal guaranteed grasp quality measure is:

$$Q_{max} = \max_{\theta} \left(\min_{i} \left(\max_{s} Q(i, \theta, s) \right) \right). \tag{3.18}$$

The best grasp will be found at the value of θ producing Q_{max}, and the contacts will be placed at the corresponding optimal values of s. The actual quality of the resulting grasp will in general be greater than Q_{max}, but it is guaranteed never to be less than Q_{max}.

3.6.8 Discussion

This section showed how to construct a generalized grasp prototype as a collection of contact constraint sets. It also showed how to define contact quality measures based on the generalized grasp prototype definition, and it showed how to find robust or high quality grasps based on these contact quality measures. These techniques are illustrated by examples in the next chapter.

There are a number of issues that merit further discussion. This section:

- reviews the reason for the constraints imposed on the grasp quality measure in Chapter 2,
- discusses the idea of defining contact constraint sets or contact regions in a different, more equitable way, and
- estimates the change in efficiency of this algorithm over the brute force algorithm of Section 3.4.

Constraints on the Grasp Wrench Space

In Chapter 2, the grasp wrench space was limited to the space spanned by the convex hull of the contact wrenches. This section makes it clear why this representation was used. All of the constructions used to define contact constraint sets rely on the assumption that the grasp wrench space is formed from the convex hull of the contact wrenches.

Adjusting Wrench Space Regions

Section 3.6.3 postponed discussion of the idea that a generalized grasp prototype might be optimized for a particular task wrench space by adjusting the contact constraint sets to maximize the sizes of the independent wrench space regions. Two brief notes on that idea are stated here:

1. The appropriate way to measure the effectiveness of such an adjustment would be to measure the sizes of independent contact regions on the target object, not the independent wrench space regions of the generalized grasp prototype. Target object geometry affects how the size of a wrench space region will map to size of an accessible region of contact on the target object, which is what we really want to maximize. The generality of the generalized grasp prototype would be sacrificed by requiring that it be readjusted for every new target object.

2. The distribution of contacts in a grasp may reflect something about an intended task that is not reflected in the shape of the task wrench space, especially if the task wrench space in use is a crude approximation, such as a wrench space ball. Altering contact constraint sets is equivalent to changing the contact distribution. If we are not certain that the task wrench space we are using exactly captures the demands of a task, such alteration may have undesirable effects.

In this report the definitions of contact constraint sets are not adjusted to maximize wrench space region size or independent contact region size.

Search Space Size

Both the brute force grasp synthesis algorithm presented in Section 3.4 and the generalized grasp prototype algorithm presented in this section require (in a naive implementation) exhaustive search through their respective quality spaces. Where the brute force algorithm looks for a large continuous region of good grasp quality measures, the generalized grasp prototype algorithm looks for an alignment of the grasp prototype to the target object that has large continuous regions of good contact quality measures for all contacts of the grasp (Expression 3.17 above). Where the brute force technique optimizes the grasp quality measure over the entire space, the generalized grasp prototype algorithm finds an alignment of the grasp prototype to the target object that maximizes the minimum optimal contact quality measure over the contacts of the grasp (Expression 3.18 above).

Although the computation required to derive contact quality is much simpler than that required to derive grasp quality, the real difference between the two algorithms is in search space size. The size of the contact quality space of the grasp synthesis algorithm of this section is cp sampled over S^1. This can be compared to the size of the contact quality space in the brute force grasp quality calculation of $\frac{p!}{c!(p-c)!}$. Because a very conservative description of the set of grasps that have high grasp quality measures has been used, lower bound grasp quality measures can be computed at each contact of the grasp. This pushes the combinatorics of contact selection inherent in the brute force approach to the much simpler problem of selecting combinations of contacts from the contact quality spaces computed in the grasp prototype approach. This is a much easier task, because *any* combination of contacts with contact quality measures above a given grasp quality threshold will work.

3.7 Working with 3D Objects

When we move from two-dimensional to three-dimensional objects, the techniques used to construct contact constraint sets and contact quality measures and to synthesize optimal grasps from these constructs do not change. The contact types and the required intermediate constructs do become more complex, however. Contact types used for three-dimensional objects were discussed in Chapter 2, and Figure 2.23 shows two of them. Important changes in the dimensionality of intermediate constructs are listed below:

- the wrench space is six-dimensional,
- there is a two-dimensional object boundary surface rather than a one-dimensional object boundary curve,
- the grasp prototype must be aligned with the target object in the three-dimensional orientation space $SO(3)$ rather than the one-dimensional orientation space S^1.

This means that constructing the contact quality space requires cp^2 calculations sampled over $SO(3)$ rather than cp calculations over S^1. Every point on the object boundary surface has a contact quality value for each contact of the grasp and for each orientation of the grasp prototype with respect to the target object. Exhaustive grasp optimization can be done exactly as before, optimizing the size of continuous contact regions or optimizing the contact quality measure, but Expressions 3.17 and 3.18 will need to be adjusted to accommodate the increased dimensionality of the alignment parameter θ (now a three-dimensional orientation parameter) and object boundary parameter s (now a two-dimensional surface).

The increased complexity of this optimization problem makes it more profitable when synthesizing grasps of three-dimensional objects to rely more on an initial geometric fit of the grasp prototype to the target object, as searching the entire space for the best alignment is very expensive. It is important to note, however, that this geometric matching process does not provide a completely well-defined starting point for a search when either the target object or the grasp prototype has any orientation symmetry (a cylinder, for example, has a one-dimensional orientation symmetry). There is no way to geometrically resolve these symmetries without added information about the target object or the environment. This problem is discussed in the second half of this report.

3.8 Summary

This chapter presented a novel technique for synthesizing grasps by generalizing a single example grasp, or grasp prototype, and applying this generalized prototype to a new target object. The method used for generalizing the prototype was shown to decrease the complexity of the search for a good grasp of the target object by allowing the contacts of a proposed grasp to be evaluated and even optimized independently. If independent contact quality measures are all above some quality threshold, then the quality of the resulting

grasp is guaranteed to also be above that threshold. Some features of the grasp synthesis technique presented in this chapter are reviewed below.

Global Region Information: The technique of finding grasps that match a generalized grasp prototype provides lower bound quality values over a space of possible grasps. This region information is important, because it can be used to determine the robustness of a grasp to errors in contact placement. It also provides the flexibility in contact placement that is needed to adapt to the constraints of the robot hand kinematics.

Offline Computation: This technique allows a grasp to be optimized for a particular task offline and then generalized explicitly for the given task wrench space. The generalized grasp prototype for that given task can then be stored in a grasp library for later access. This means that any given grasp design and task description can incorporate knowledge gained from extensive analysis and experimentation.

Speed: No matter how complex and time-consuming the offline grasp design and analysis may be, the runtime process of testing potential grasps can be very fast. It is reduced from the problem of constructing a grasp wrench space convex hull and finding the scale of the largest task wrench space that fits within that convex hull to the problem of evaluating a few dot products for each contact of the grasp. The quality measure for each contact can be independently calculated, thus allowing the placement of each contact to be independently optimized. This greatly reduces the combinatorics of the problem of finding a high-quality set of contacts on a target object.

Chapter 4

Examples Using 2D Objects

This chapter shows how the constructs of the previous chapter, in particular the contact constraint sets and the contact quality measures, can be visualized. It also demonstrates the use of these constructs in synthesizing both robust grasps and optimal quality grasps of new target objects.

A single grasp prototype is used throughout this chapter. This grasp prototype is shown in Figure 4.1. Grasp quality is computed as in Equation 2.25, where the task wrench space is assumed to be a wrench space ball. Section 4.1 addresses the problem of visualizing the contact constraint sets of the example grasp prototype for the given task wrench space, and Section 4.2 demonstrates the synthesis of robust grasps of the four new target objects in Figure 4.2 using these contact constraint sets. Section 4.3 shows how the contact quality measures of the example grasp prototype can be visualized, and Section 4.4 demonstrates the synthesis of optimal quality grasps of the four target objects in Figure 4.2. Section 4.5 presents a summary.

4.1 Visualizing Contact Constraint Sets

This section addresses the problem of visualizing the contact constraint sets of the grasp prototype shown in Figure 4.1. Some examples of contact constraint sets have already been presented in the previous chapter (for example, Figure 3.13). In these examples, the entire wrench space was shown, with the contact constraint sets as shaded regions within that wrench space. This was possible because the examples involved only a two-dimensional wrench space. In general, however, two-dimensional objects have a three-dimensional wrench space, $[f_x \quad f_y \quad \tau_z]^T$, and so it is useful to move to a more efficient format for illustrating the contact constraint sets. In particular, the interesting portions of the contact constraint sets are really two-dimensional. We are interested in the contact wrenches that could be used to construct the grasp wrench space of a grasp. The grasp wrench space of any grasp is bounded by the convex hull of the contact wrenches resulting from unit applied forces at the contacts of the grasp. Because the applied forces at the contacts of the grasp are normalized, there are only two degrees of freedom available with

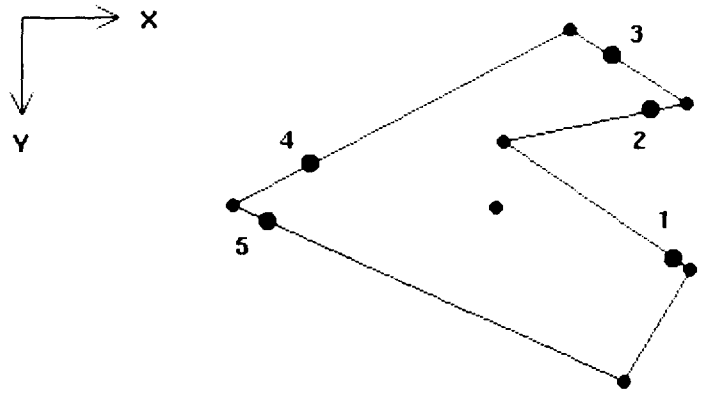


Figure 4.1: An example grasp prototype, represented by forces applied at the numbered contacts.

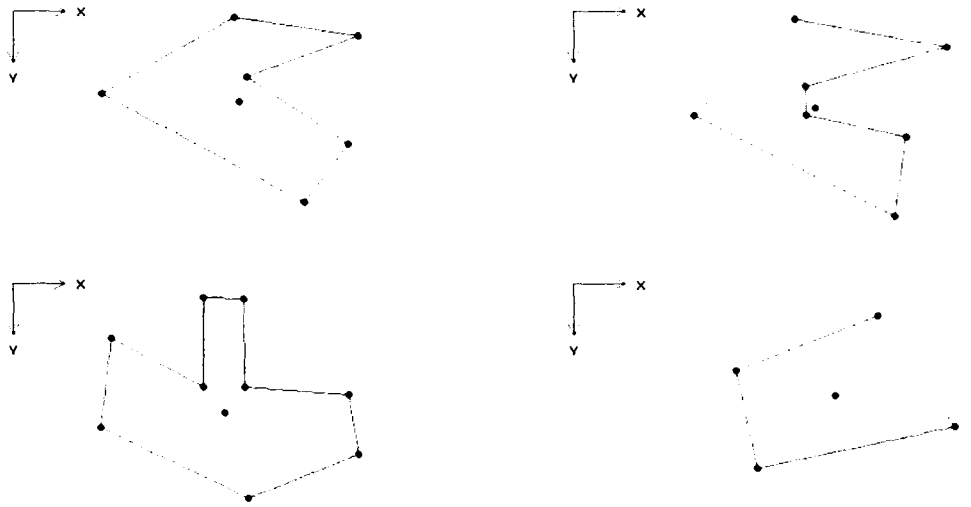


Figure 4.2: This chapter will demonstrate synthesis of optimal grasps of these four target objects. The objects were designed to have varying degrees of similarity to the example target object.

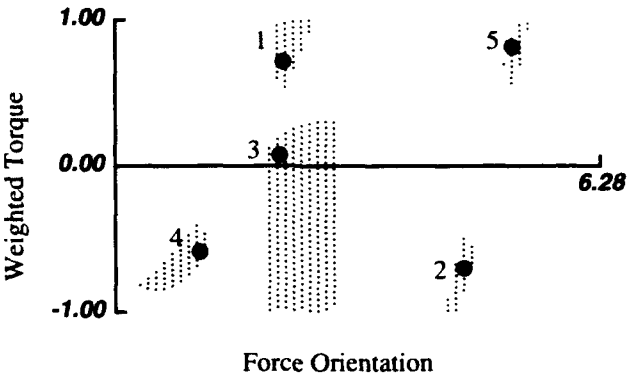


Figure 4.3: Independent wrench space regions for the five contacts of the example grasp that guarantee a grasp quality measure of 75% of the original grasp quality measure. These regions of normalized contact wrenches are presented in a space of force orientation vs. weighted torque values. The normalized wrenches corresponding to the five contacts of the original grasp are shown.

which to specify the space of possible contact wrenches. These degrees of freedom can be captured, for example, with an applied force orientation and a torque magnitude.

Figure 4.3 shows a plot of the contact constraint sets of the example grasp in Figure 4.1. Each point in this plot is a contact wrench. The x-axis represents orientation of the force vector in radians, and the y-axis represents the scaled torque vector. The shaded regions illustrate wrenches having contact quality measures of at least 75% of the grasp quality measure of the example grasp. The numbered points within these regions represent the contact wrenches of the example grasp. Each contact wrench falls, of course, within its corresponding contact constraint set.

The plot used to illustrate contact constraint sets is the same plot as that used to display the complete set of contact wrenches that could be applied to the target object, the object boundary curve in Figure 2.12. Figure 4.4 shows the object boundary curve overlaid on the plot of contact constraint sets. Recall that the vertical lines represent contact with edges of the target object, since the orientation of applied force does not vary as the contact point is moved along an edge. Curved lines represent contact with vertices of the target object, since the torque varies in a nonlinear way with the orientation of the edge contacting the vertex. The point-edge contacts of the example grasp fall on the vertical lines of the object boundary curve.

The combined plot of object boundary curve and contact constraint sets tells us a great deal about the robustness of a grasp to errors in contact placement. From this plot we see, for example, that as long as point contacts are placed within the highlighted regions of Figure 4.5, the resulting grasp will have a grasp quality measure of at least 75% of the example grasp. Note that the contact region corresponding to contact 3 is discontinuous; this region overflows onto the edge containing contact 1. This can be seen

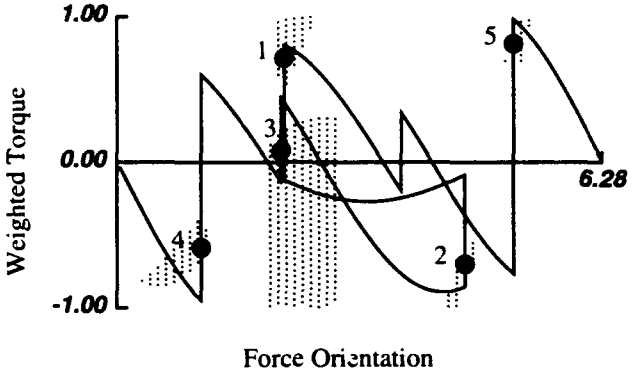


Figure 4.4: Independent wrench space regions superimposed on the object boundary curve of the example polygon. The object boundary curve indicates what contact wrenches are possible for this object.

in Figure 4.4 as well: there are two nearly overlapping vertical lines intersecting region 3, which corresponds to contact regions on two separate edges of the target object. Also note that contact regions may involve not only point-edge contacts, but also edge-vertex contacts. For example, Figure 4.4 shows that the contact constraint set of contact 4 can be satisfied by one of two discontinuous contact regions: the region on the edge of the target object shown in Figure 4.5, or a separate region of edge contact on the nearby target object vertex.

The effect due to a variety of changes in the grasp can be read from the plot in Figure 4.4. For example, uncertainty in contact placement displaces contact points on the object boundary curve. In addition, uncertainty in object orientation causes the curve to slide on the orientation axis, and changes in object geometry cause the object boundary curve to warp. As edge lengths shrink, the vertical lines shrink. As edge orientations change, these lines move horizontally. We can get an idea of how robust a grasp might be by noting how easily such changes can cause the contact points to slip out of the good contact regions.

This leads to the question of object design. In other words, what changes in target object geometry would keep some part of the object boundary curve within the given contact constraint sets? One way to visualize a partial answer to this question is to sample wrenches in the contact constraint sets and translate them into edge segments, such as the highlighted segments in Figure 4.5. Point contact on these edge segments would generate contact wrenches within the given constraint sets. Figures 4.6 and 4.7, for example, show sampled sets of edge segments corresponding to the contact constraint sets for contacts 1 and 4 of the example grasp. A new target object with an edge covering part of any of these segments would contain a range of point-edge contacts satisfying the contact constraint sets of either contact 1 or contact 4. This representation is, of course, not a complete representation of the contact constraint sets of contacts 1 and 4. Aside

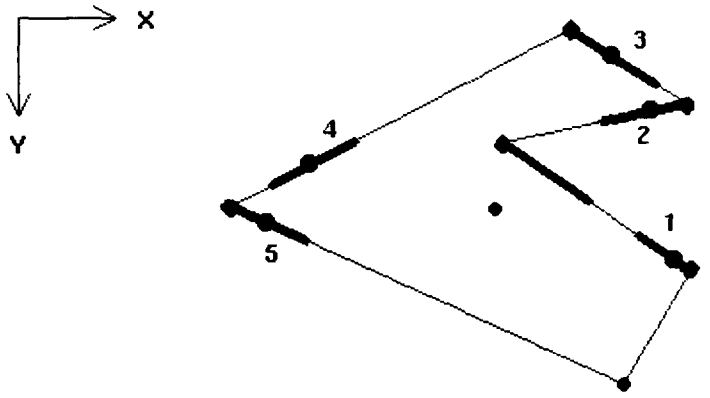


Figure 4.5: Independent contact regions that guarantee a grasp quality measure of 75% of the grasp quality measure of the original grasp. Only point-edge contacts are shown. The region corresponding to contact three overflows onto the edge containing contact one.

from the fact that the edge segments are sampled, only point contact with potential object edges is represented. Possible edge contacts with object vertices are not shown.

The real problem addressed by this report is not object design, however, but the problem of finding the best fit of a grasp prototype to an existing target object. A technique for synthesizing robust grasps of a given target object will be illustrated in the next section.

4.2 Finding Robust Grasps

The previous section showed how the contact constraint sets of a generalized grasp prototype can be visualized. This section shows how these regions can be used to design grasps for target objects that differ from the prototype object. Although the calculations that will be shown below could be simplified by beginning with a geometric fit of a new target object to the prototype object, with two-dimensional objects it is more interesting to plot results for the entire space of possible matches.

We begin with the example polygon. The fit of this polygon to a generalized grasp prototype can be measured by computing the sizes of the independent contact regions that result. Contact region sizes can be approximated from the lengths of object boundary curve that pass through the occupied areas in Figure 4.4. To make this approximation work, a parameter that reasonably compares distance in the torque dimension with distance in the orientation dimension is needed. In the experiments below, the total torque span of 2 is considered to be a distance equivalent to the orientation span of 2π.

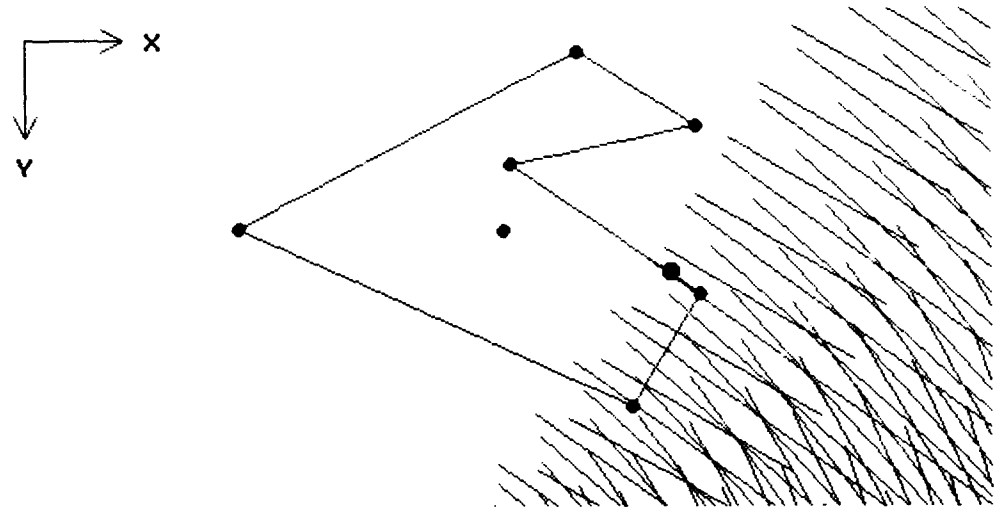


Figure 4.6: Sampling of legal contact segments for contact 1. Each segment could be a good region of point-edge contact on a new target object with an edge containing that segment.

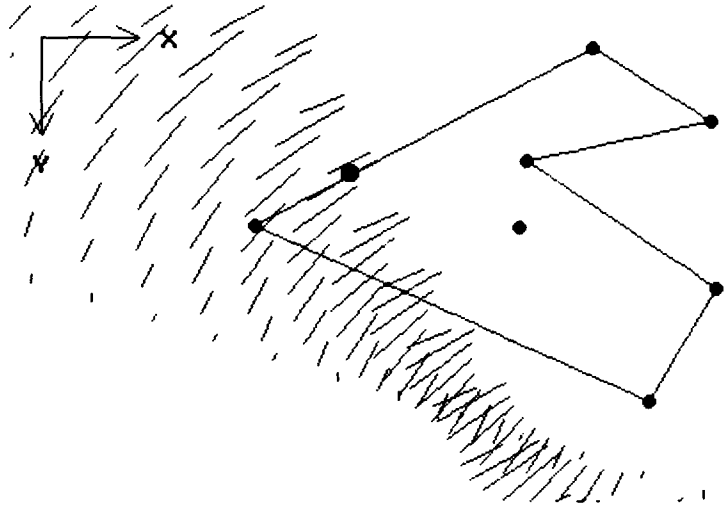


Figure 4.7: Sampling of legal contact segments for contact 4. Each segment could be a good region of point-edge contact on a new target object with an edge containing that segment.

There is one degree of freedom in the match of a two-dimensional object to a grasp prototype. This degree of freedom is the relative orientation of the target object with respect to the prototype (refer to Expression 3.17). The matches that result as this orientation is varied can be checked by sliding the boundary curve of the object across the plot in Figure 4.4 while the legal wrench space regions remain constant.

Figure 4.9 shows the results of performing this operation. Region sizes were computed that guarantee quality measures of 75%, 50%, and 25% of the example grasp quality measure (see Figure 4.8). The upper plot of Figure 4.9 shows how the minimum contact region sizes vary with orientation of the example polygon. For the most part, matches are limited to a small region of orientations near the original solution. There seems to be an additional match at an orientation of approximately π, however. The lower plot shows this match. The 25% quality regions are also displayed in the figure, and we can see that the match is extremely fragile. A small positive change in orientation will push the curve out of region 5. A small negative change in orientation will push the curve out of region 2.

There is a significant amount of structure contained in a grasp prototype. We can see something of how this will affect the range of possible matches to a prototype by performing some simple experiments.

Figures 4.10 through 4.13 show some results for the four objects shown in Figure 4.2. The objects show decreasing similarity to the example polygon. Minimum contact region sizes were computed for each object that guarantee grasp quality measures of 75%, 50%, and 25% of the example grasp quality measure. The middle plot of each figure shows how the minimum contact region size varies with object orientation. While the first two objects show a reasonably robust match to the grasp prototype about an orientation of zero, this match is more questionable with the third object, and completely absent in the fourth object.

As something of a control, the matches of these objects to a symmetrical five-contact prototype were also computed. Regions for this prototype that guarantee quality measures of 75%, 50%, and 25% of the example grasp quality measure are shown in Figure 4.14. Compare these to the corresponding prototype regions in Figure 4.8. The matches of the new objects to this symmetrical prototype are shown in the lower plots of Figures 4.10 through 4.13. Some matches are evident here, particularly in the 5.5 to 6 radian range, but the nicely shaped curves due to the original prototype are absent. The degradation of these matches is sharp, not graceful, because of the shapes of the contact constraint sets of the symmetrical grasp prototype.

A robust grasp can be found for each new object using the central plots in Figures 4.10 through 4.13. Figures 4.15 through 4.18 show the results of this process. For each example, a good shift in orientation was chosen from the central, minimum region size plot in each of Figures 4.10 through 4.13. The shifted object boundary curve was plotted against the 25% quality regions of the original prototype. These plots are shown in the bottom halves of the figures. Using these plots, reasonably robust contacts were chosen and grasp quality was calculated. These grasps and quality measures are shown in the top halves of Figures 4.15 through 4.18.

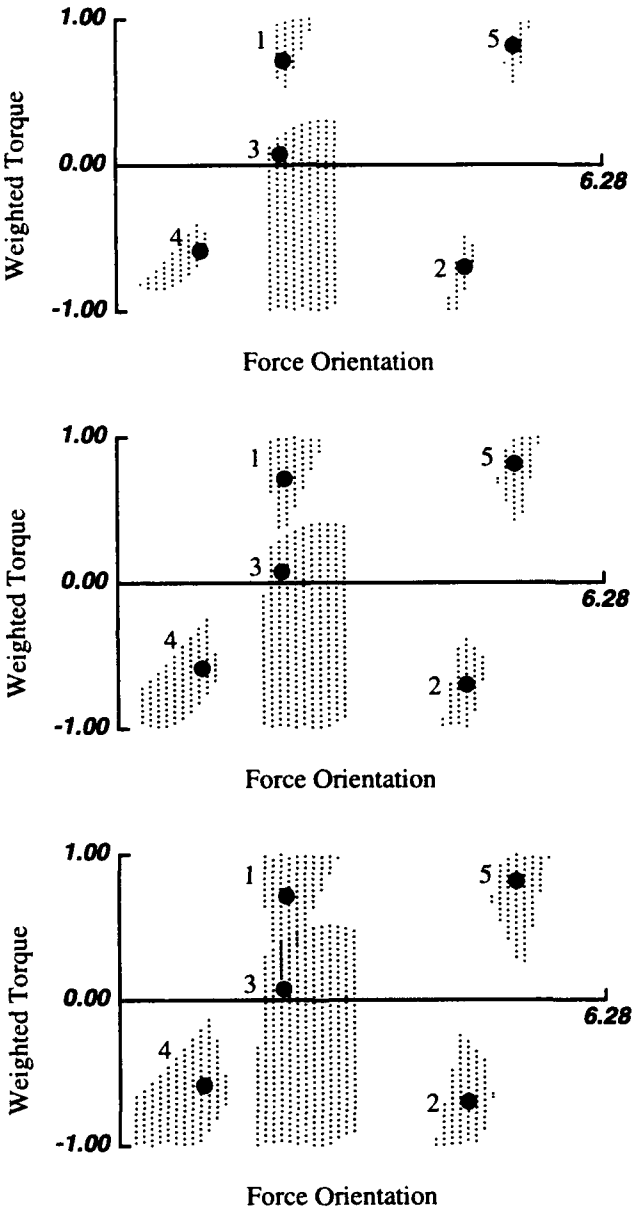


Figure 4.8: Independent wrench space regions for the example grasp. The regions shown are for qualities of 0.2625, 0.175, and 0.0875, or 75%, 50%, and 25% of the grasp quality measure of the example grasp.

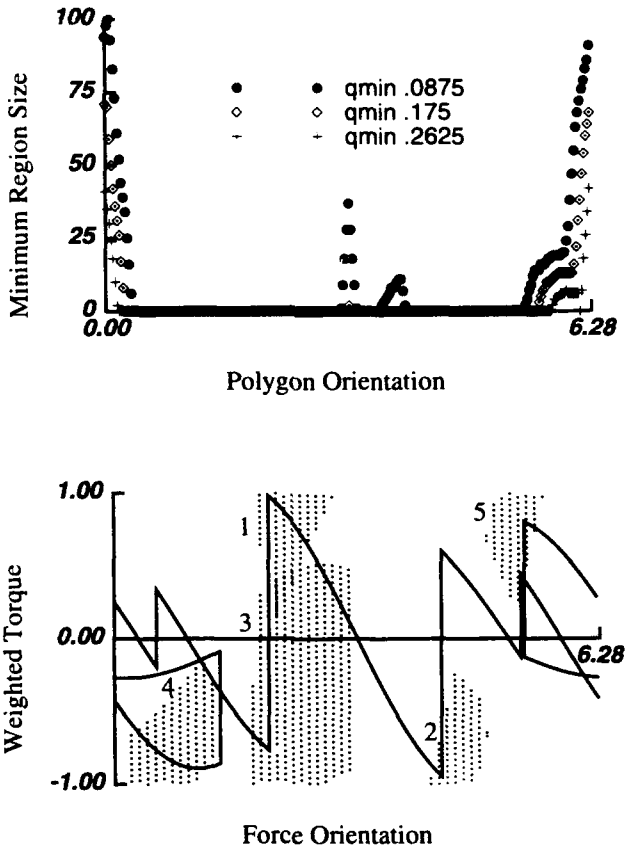


Figure 4.9: The upper plot shows the minimum contact region size on the example polygon for three grasp quality thresholds vs. the relative orientation of the grasp prototype with respect to the target object. The best match falls at a relative orientation of zero. The lower plot shows the marginal match found at an orientation of π, superimposed on the 0.0875 quality regions.

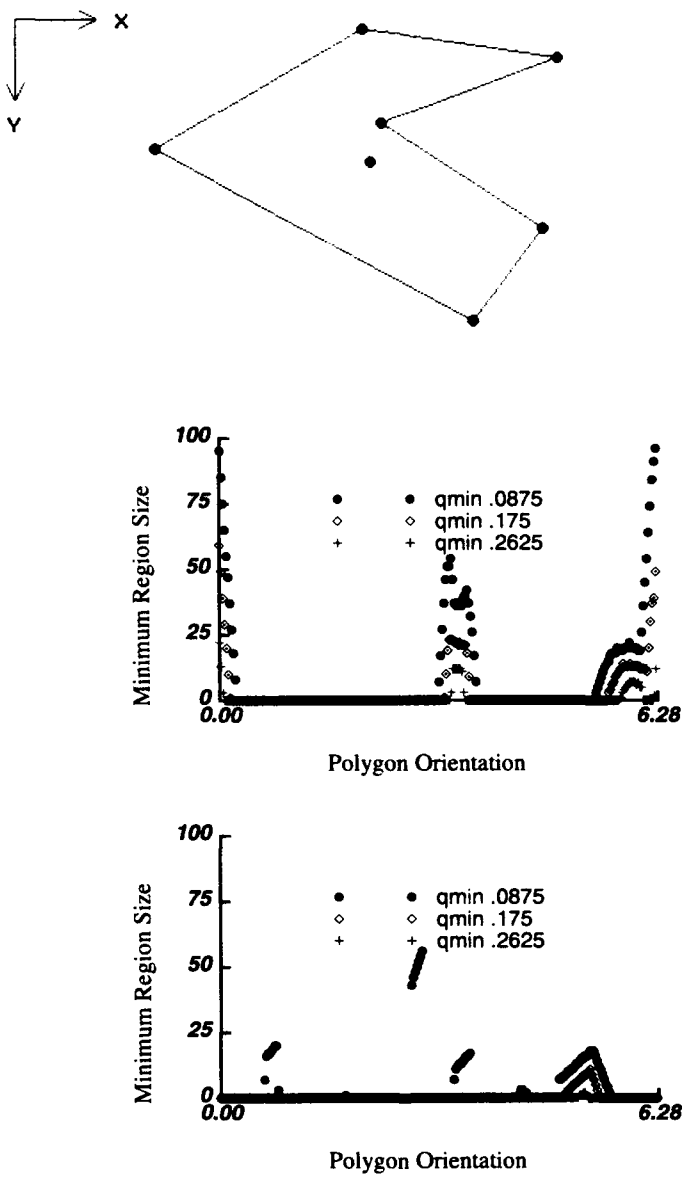


Figure 4.10: New object 1 matches to the example polygon grasp prototype. The top figure shows new object 1. The two plots show the minimum contact region size on new object 1 for three grasp quality thresholds vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space regions derived from the example grasp (Figure 4.8). The bottom plot uses the symmetrical wrench space regions of the control grasp (Figure 4.14).

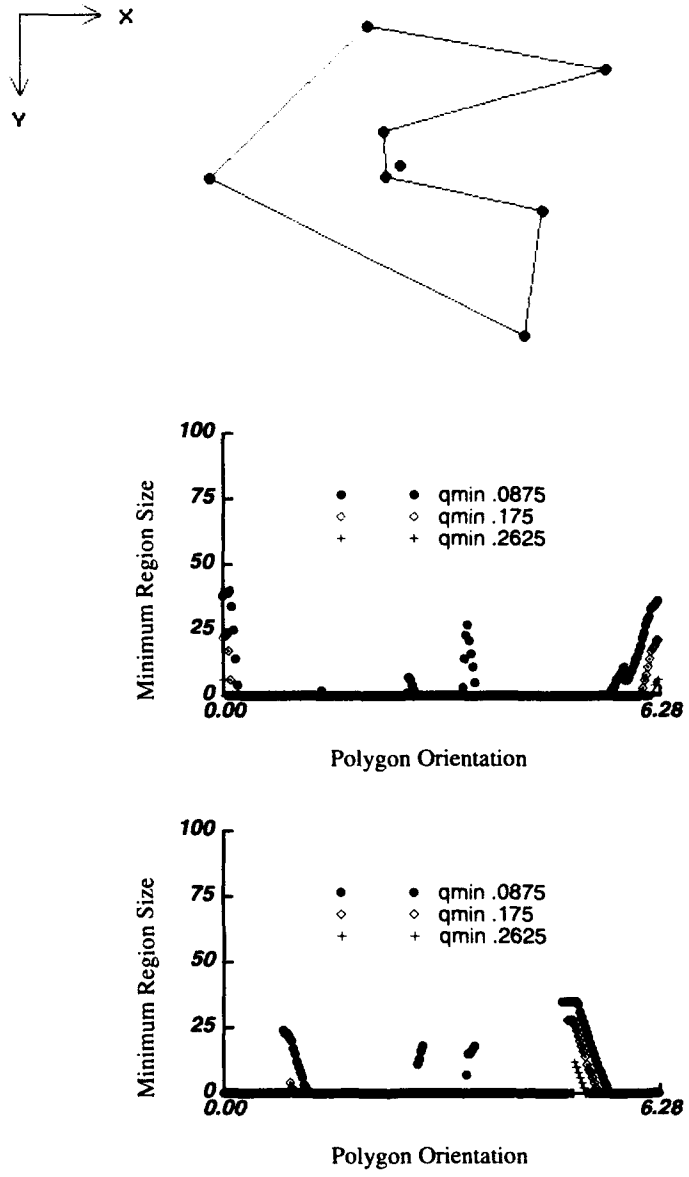


Figure 4.11: New object 2 matches to the example polygon grasp prototype. The top figure shows new object 2. The two plots show the minimum contact region size on new object 2 for three grasp quality thresholds vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space regions derived from the example grasp (Figure 4.8). The bottom plot uses the symmetrical wrench space regions of the control grasp (Figure 4.14).

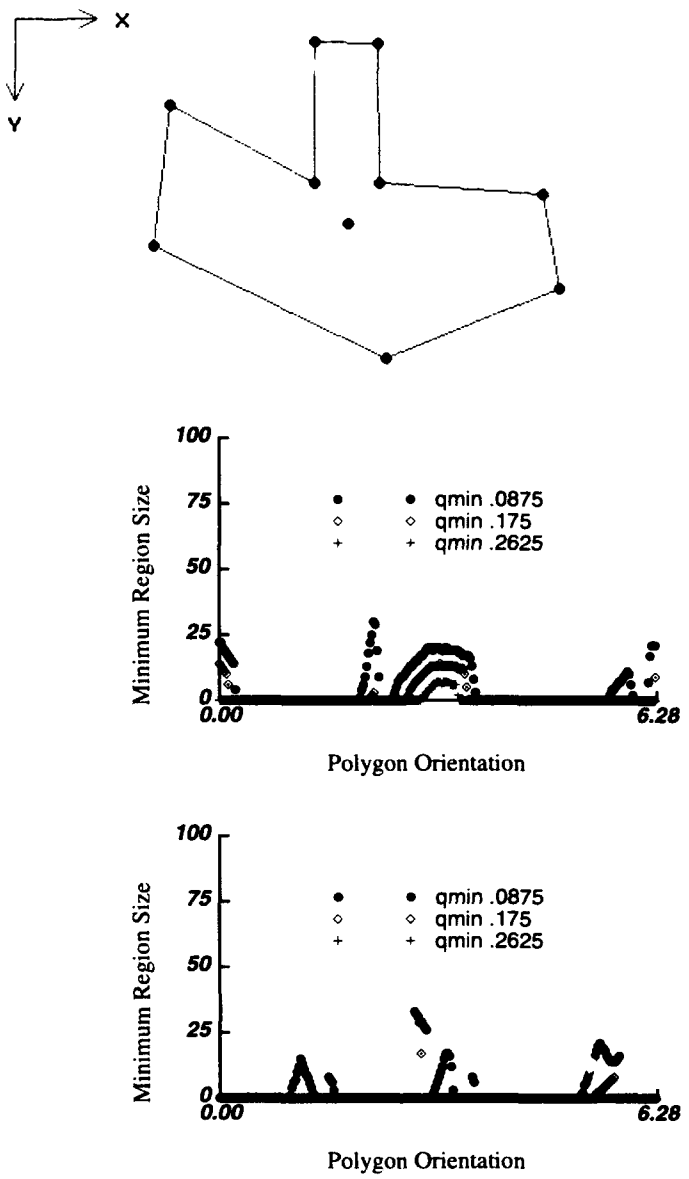


Figure 4.12: New object 3 matches to the example polygon grasp prototype. The top figure shows new object 3. The two plots show the minimum contact region size on new object 3 for three grasp quality thresholds vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space regions derived from the example grasp (Figure 4.8). The bottom plot uses the symmetrical wrench space regions of the control grasp (Figure 4.14).

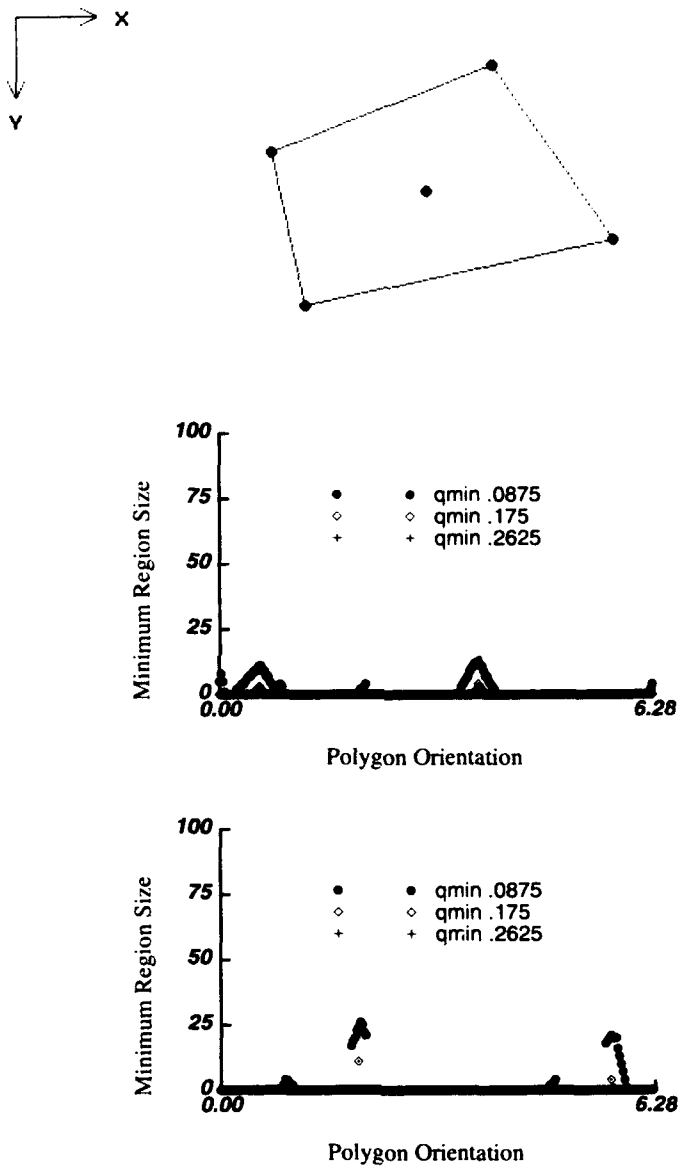


Figure 4.13: New object 4 matches to the example polygon grasp prototype. The top figure shows new object 4. The two plots show the minimum contact region size on new object 4 for three grasp quality thresholds vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space regions derived from the example grasp (Figure 4.8). The bottom plot uses the symmetrical wrench space regions of the control grasp (Figure 4.14).

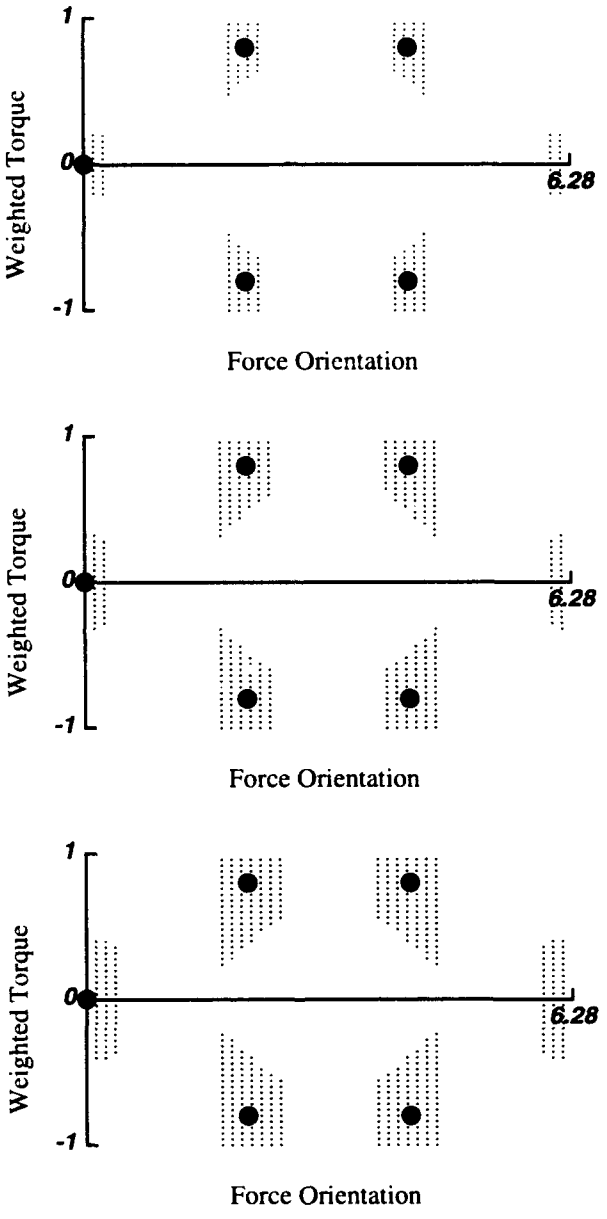


Figure 4.14: Independent wrench space regions for a symmetrical control grasp. The regions shown are for qualities of 0.2625, 0.175, and 0.0875, to parallel the 75%, 50%, and 25% regions of the example grasp shown in Figure 4.8.

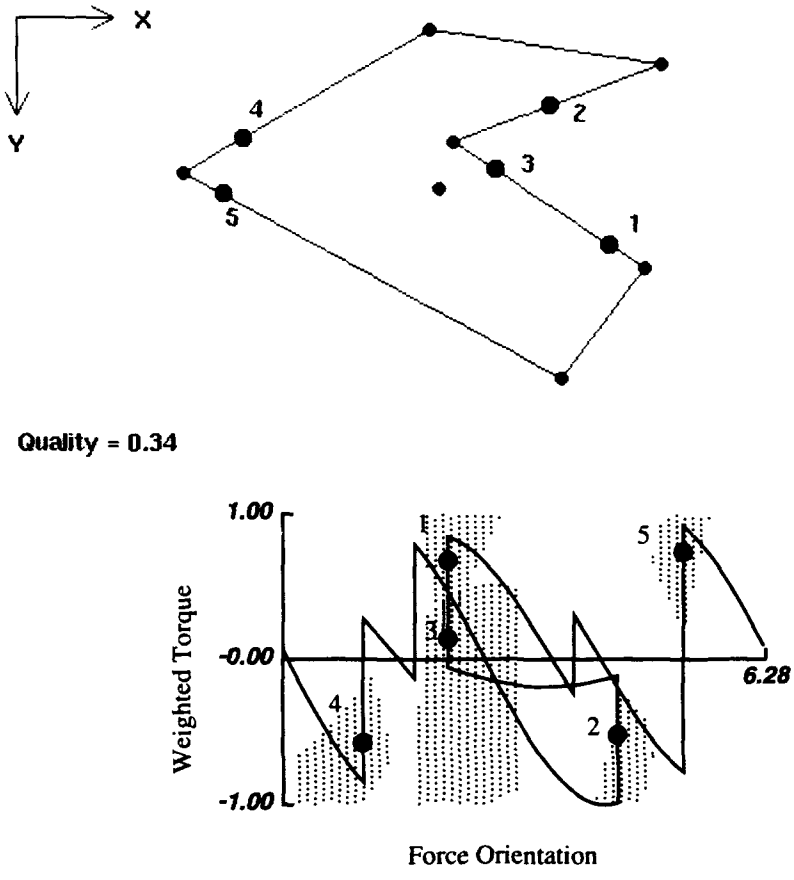


Figure 4.15: New object 1 best match to the example grasp has a grasp quality measure of 0.34. In the plot, the object boundary curve of new object 1 has been superimposed on the 0.0875 quality regions. This object boundary curve has not been shifted with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

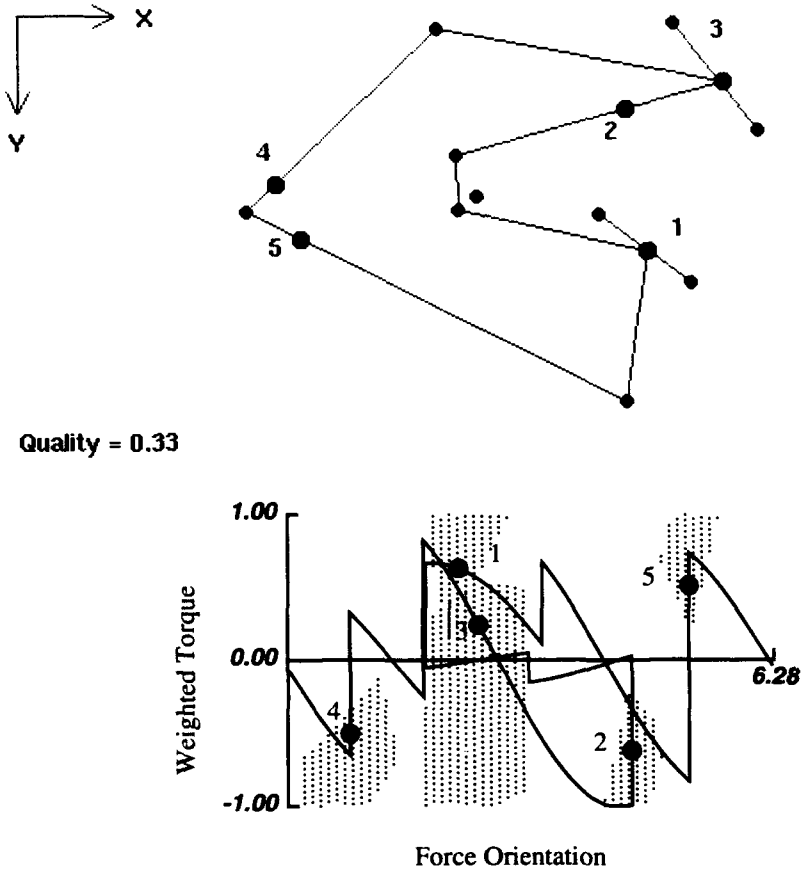


Figure 4.16: New object 2 best match to the example grasp has a grasp quality measure of 0.33. In the plot, the object boundary curve of new object 2 has been superimposed on the 0.0875 quality regions. This object boundary curve has not been shifted with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

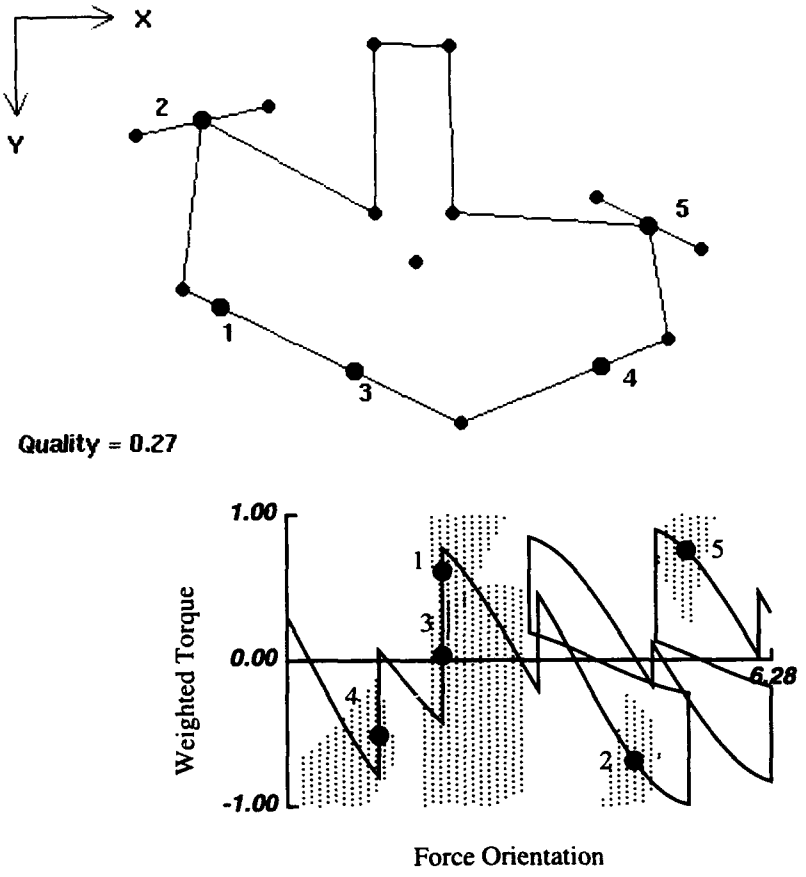


Figure 4.17: New object 3 best match to the example grasp has a grasp quality measure of 0.27. In the plot, the object boundary curve of new object 3 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by π with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

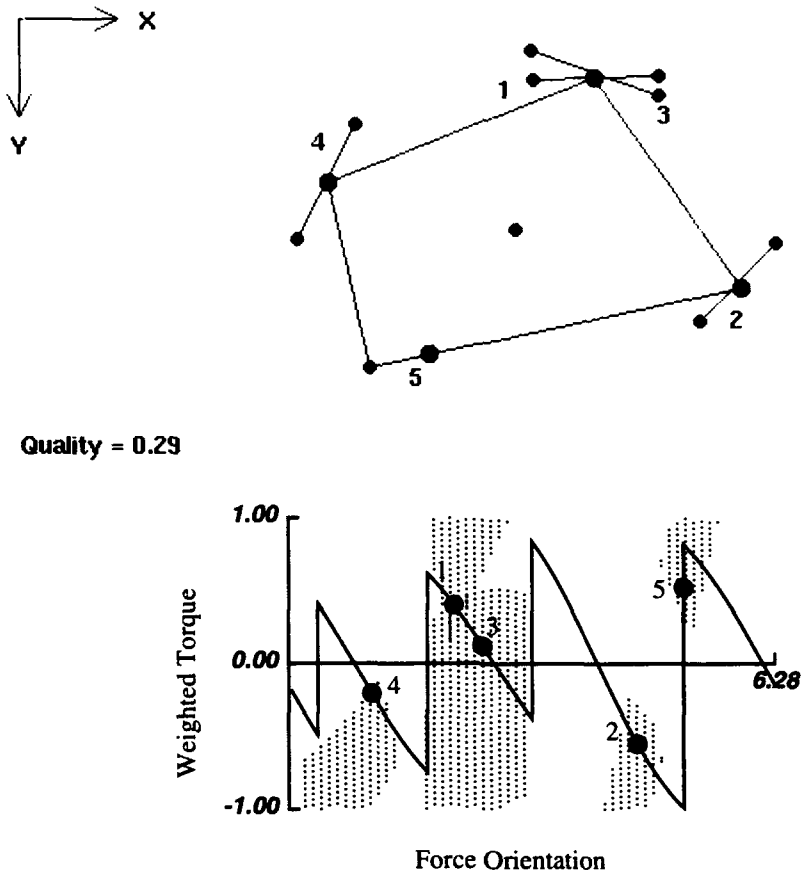


Figure 4.18: New object 4 best match to the example grasp has a grasp quality measure of 0.29. In the plot, the object boundary curve of new object 4 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by 0.6 with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

Note that the qualities of all four grasps are much better than the guaranteed lower bound of 0.0875. They are very near the quality of the grasp prototype, which is 0.35. Also note, however, that there is a tendency to resort to vertex contacts as the object becomes less like the prototype. The grasp in Figure 4.18, in fact, cannot be achieved at all with frictionless hard finger contacts. It requires that two contacts at different orientations be placed on the same vertex.

4.3 Visualizing the Contact Quality Measure

The previous sections showed how contact constraint sets can be visualized, and they presented some examples of optimizing the sizes of independent contact regions for a given grasp quality threshold. Contact region size is important because it provides an indication of the robustness of a grasp to small errors in contact placement. Grasps can also be optimized using the contact quality measure, however. Each contact quality measure provides a lower bound on the quality measure of the grasp as a whole. Optimizing contact quality measures individually can improve on these lower bounds and thus improve the quality measure guaranteed for the grasp.

The contact quality measure is a measure of the greatest grasp quality threshold for which a given contact wrench would fall within a contact constraint set of a generalized *grasp prototype*. To visualize contact quality measures means to show how the wrench space described by the contact constraint sets varies with the grasp quality threshold.

The wrench space regions that satisfied particular contact constraint sets were displayed in Sections 4.1 and 4.2 by plotting these regions in a space of force orientation vs. weighted torque (e.g. Figure 4.8). The region sizes varied with the grasp quality threshold used to define the contact constraint sets. Lower grasp quality thresholds created a larger space of contact wrenches for each contact, and this showed up as larger regions in the wrench space plots. Some information can be added to these contact constraint set plots to obtain a coarse representation of the contact quality measures. Figure 4.19 shows contact quality plots for both the prototype grasp (top) and the symmetrical grasp (bottom) of the previous sections. The contact regions in these plots are striped at contact quality intervals of 0.1, with the outer, black stripe representing contact quality measures 0 to 0.1, followed by a white stripe representing contact quality measures 0.1 to 0.2 and so on. The largest region in the top figure has contact quality measures greater than 0.6 in its center.

4.4 Finding Optimal Quality Grasps

Now that one representation of the contact quality space has been shown, some results of optimizing grasps based on the contact quality measures are presented. As in Section 4.2, the symmetrical grasp prototype can be compared to the grasp prototype from the example polygon, and the section begins by plotting results for the entire space of possible matches of these grasp prototypes to the original example polygon and the four target objects

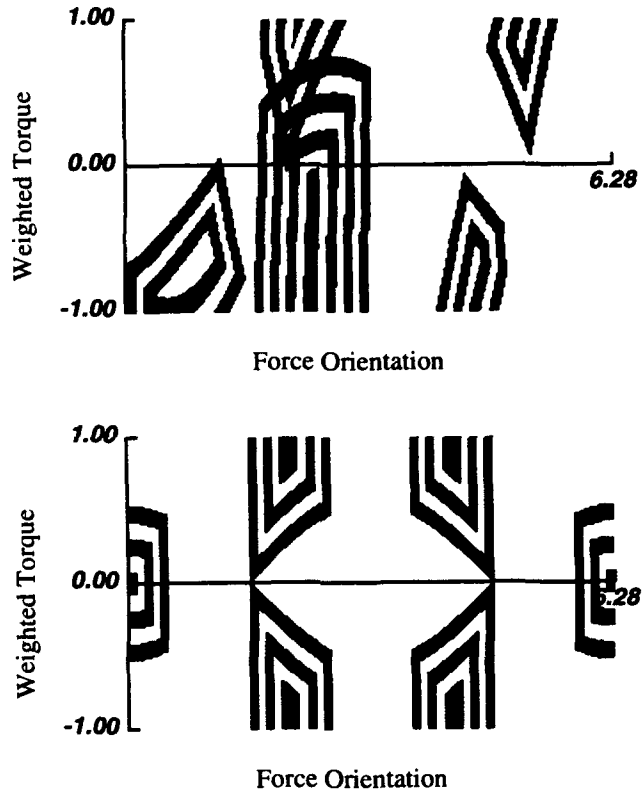


Figure 4.19: Gradient plot of independent wrench space regions for the example prototype grasp (top) and the symmetrical grasp (bottom). The regions are striped at contact quality intervals of 0.1, with the outer, black stripe representing the contact quality interval [0.0, 0.1).

shown in Figure 4.2.

Expression 3.18 is used to compute lower bound grasp quality measures. As the alignment between the grasp prototype and the target object is varied (parameter θ in Expression 3.18), the contact quality measure is maximized for each contact, and contact quality measures are minimized over the contacts of the grasp.

Figures 4.20 through 4.24 show the results. Each figure shows a target object (top), a plot of lower bound grasp quality against alignment of the example grasp prototype to the target object (middle), and a plot of lower bound grasp quality against alignment of the symmetrical prototype to the target object (bottom). In the plots, any y-axis values greater than zero represent orientations from which stable grasps can be generated. The central plots show that objects similar to the example target object tend to have large regions of positive y-axis values around an orientation of zero, although each plot shows a number of orientation ranges for which stable grasps are possible. Where the peaks of these contact quality measures go above the grasp quality thresholds used to generate the plots of Figure 4.8, non-zero region sizes are seen in the plots of Figures 4.10 through 4.13. The contact quality measure plots provide information complementary to the contact region size plots. Where the region size plots indicate the robustness of a grasp to small errors in contact placement for a given grasp quality threshold, the contact quality plots show the optimal grasp quality threshold that could be established for a given orientation.

It is possible to find the grasps that correspond to the greatest lower bound grasp quality measure, and Figures 4.26 through 4.30 show the grasps that were identified for these objects. These grasps can be compared to the grasps having optimal contact region sizes in Figure 4.1 and Figures 4.15 through 4.18. The table in Figure 4.25 compares the grasps in terms of grasp quality. This table shows the optimal lower bound grasp quality measures (column 1), the actual grasp quality measures of grasps optimized using the contact quality measures (column 2), and actual grasp quality measures of grasps optimized for contact region size (column 3).

If the grasps represented by these quality measures are examined, some similarities can be identified. The grasps of the example polygon and new objects 2 and 3 are similar, although optimizing contact quality measures tends to push contacts out to the geometric extremes of the target object (the vertices), resulting in grasps that sometimes have better quality measures (e.g. grasps of the example polygon and new object 3), but may be more difficult to achieve. Many of these grasps, for example, have contacts on or near the same vertex.

The actual grasp quality measures of the grasps optimized using the contact quality measures are not necessarily as good as those optimized for contact region size. This is the case, for example, with new object 2. This is not a contradiction, because the actual grasp quality measures are the measures formed by constructing the convex hull of the contact wrenches and scaling this convex hull to include the task wrench space, here the unit wrench space ball. Contact quality measures provide only a lower bound on the actual grasp quality measure. The grasps having optimal contact quality measures will not necessarily be the ones with the optimal grasp quality measure, as the example grasps of new object 2 demonstrate.

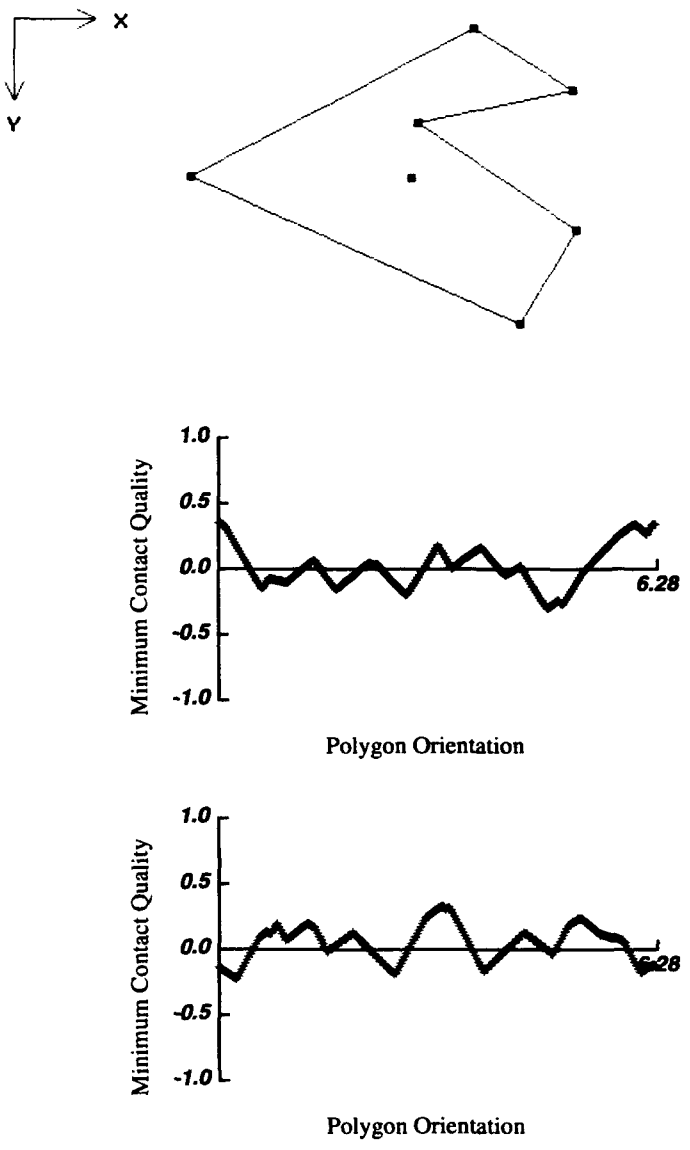


Figure 4.20: The top figure shows the example polygon. The two plots show the minimum optimal contact quality measure on the example polygon vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space gradient plot derived from the example grasp (Figure 4.19 top). The bottom plot uses the wrench space gradient plot derived from the symmetrical control grasp (Figure 4.19 bottom). Only positive contact quality measures represent stable grasps.

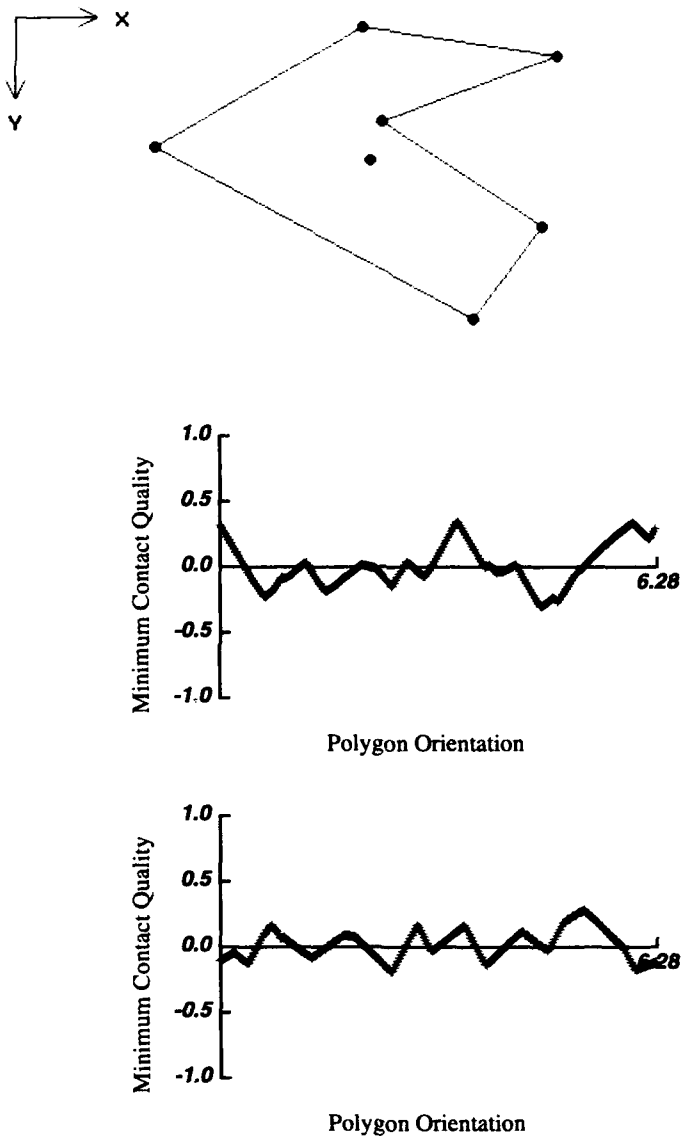


Figure 4.21: The top figure shows new object 1. The two plots show the minimum optimal contact quality measure on the example polygon vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space gradient plot derived from the example grasp (Figure 4.19 top). The bottom plot uses the wrench space gradient plot derived from the symmetrical control grasp (Figure 4.19 bottom). Only positive contact quality measures represent stable grasps.

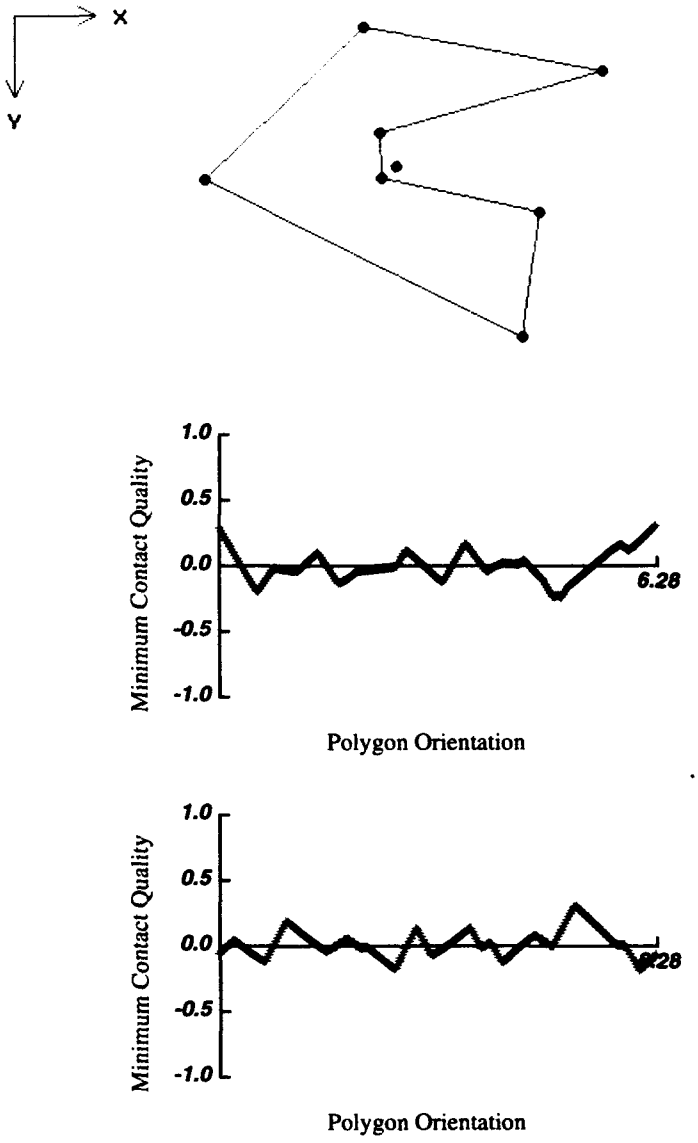


Figure 4.22: The top figure shows new object 2. The two plots show the minimum optimal contact quality measure on the example polygon vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space gradient plot derived from the example grasp (Figure 4.19 top). The bottom plot uses the wrench space gradient plot derived from the symmetrical control grasp (Figure 4.19 bottom). Only positive contact quality measures represent stable grasps.

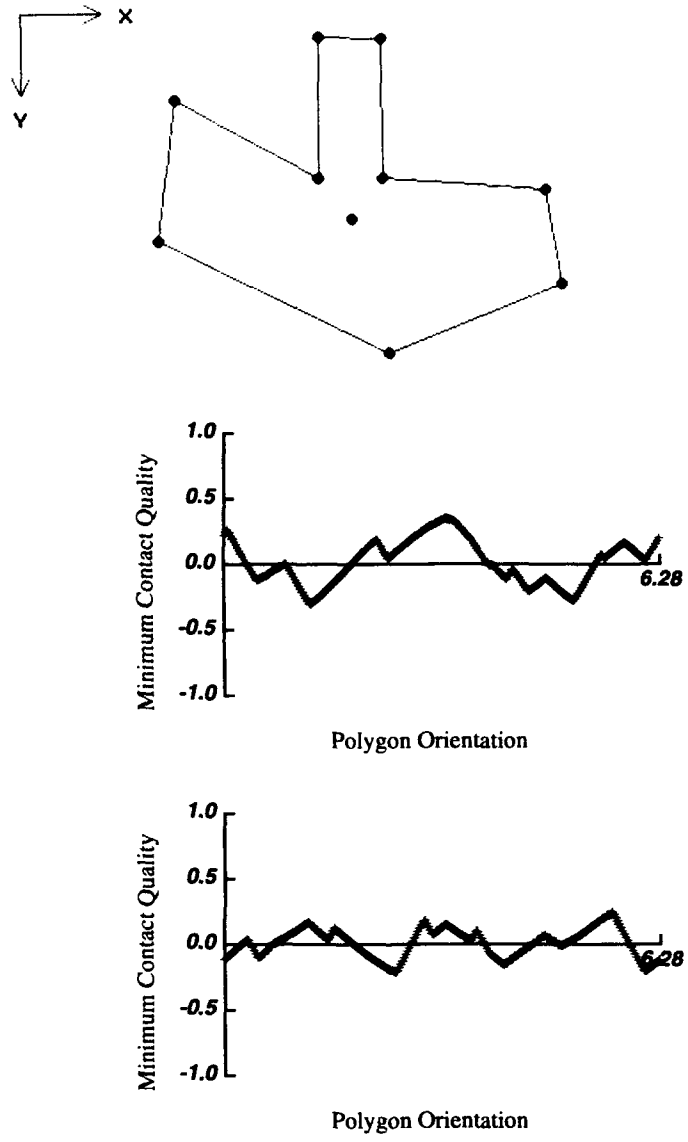


Figure 4.23: The top figure shows new object 3. The two plots show the minimum optimal contact quality measure on the example polygon vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space gradient plot derived from the example grasp (Figure 4.19 top). The bottom plot uses the wrench space gradient plot derived from the symmetrical control grasp (Figure 4.19 bottom). Only positive contact quality measures represent stable grasps.

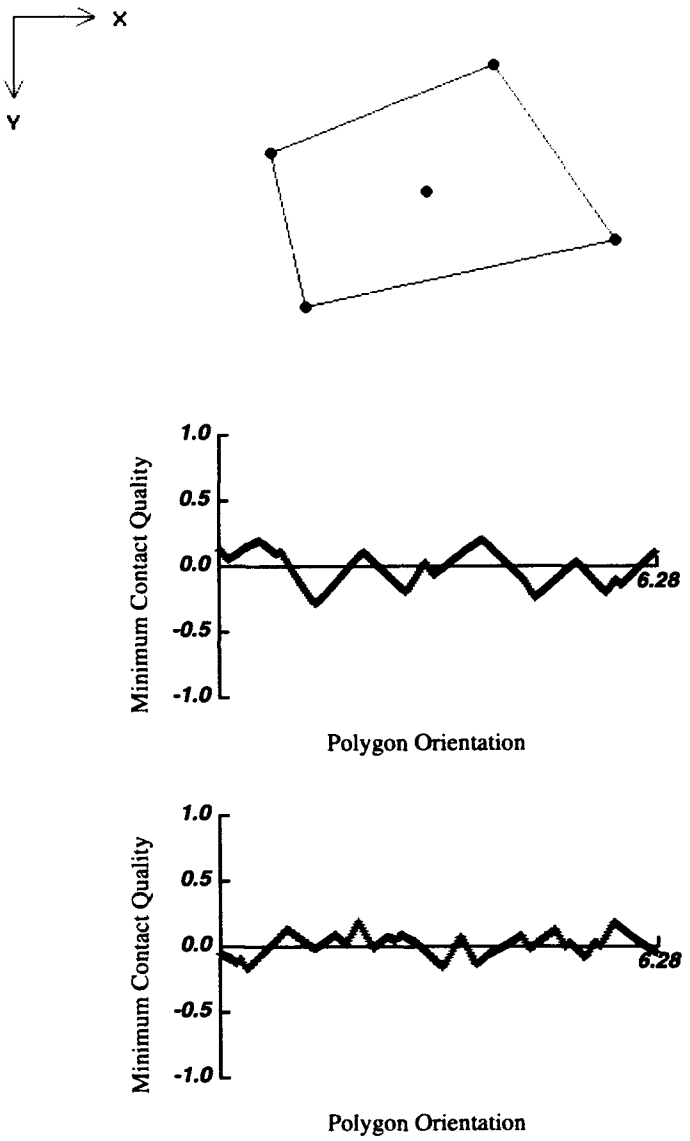


Figure 4.24: The top figure shows new object 4. The two plots show the minimum optimal contact quality measure on the example polygon vs. the relative orientation of the grasp prototype with respect to the target object. The top plot uses the wrench space gradient plot derived from the example grasp (Figure 4.19 top). The bottom plot uses the wrench space gradient plot derived from the symmetrical control grasp (Figure 4.19 bottom). Only positive contact quality measures represent stable grasps.

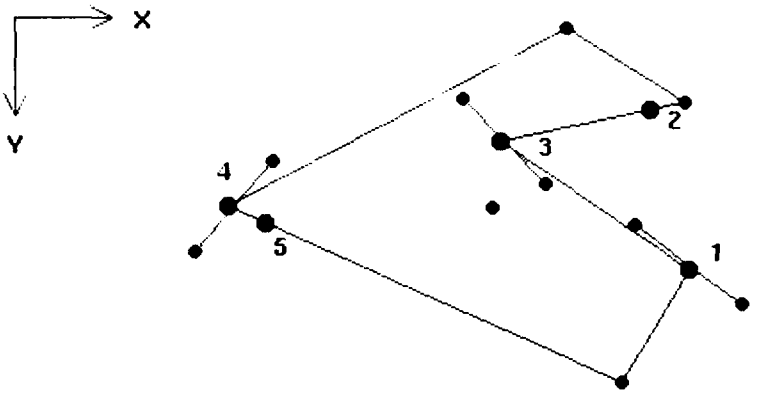
	GQ lower bound	GQ opt cq	GQ opt region_size
prototype	0.35	0.38	0.35
new object 1	0.34	0.34	0.34
new object 2	0.30	0.30	0.33
new object 3	0.35	0.36	0.27
new object 4	0.20	0.31	0.29

Figure 4.25: Table comparing the optimal lower bound grasp quality measures found by matching a prototype to a variety of target objects (column 1, taken from the top plots in Figures 4.20 through 4.24), the *actual* grasp quality measures of grasps having these optimal lower bound grasp quality measures (column 2, grasp quality measures of the grasps in Figures 4.26 through 4.30), and the actual grasp quality measures of grasps optimized for contact region size (column 3, grasp quality measures of the grasps in Figures 4.15 through 4.18.

For two of the objects, new objects 1 and 4, the grasps found by optimizing for contact quality are very different from those formed by optimizing for contact region size. To make a good selection between the two families of grasps would require consideration of both the contact quality and the region size values.

4.5 Summary

This chapter showed how contact constraint sets and contact quality measures could be visualized, and it used these constructs to form both robust grasps and high quality grasps of the example polygon and four new target objects. The examples illustrated that these grasp optimization techniques, which require only a moderate amount of computation for two-dimensional target objects, are also fairly flexible with respect to changes in target object geometry. Even when the new target objects were very different from the example object, which was used to define the grasp prototype, it was possible to find a guaranteed stable grasp of the new target object by matching it to the generalized grasp prototype.



Quality = 0.38

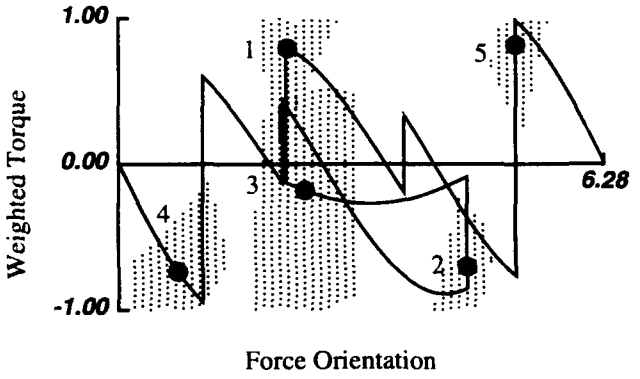
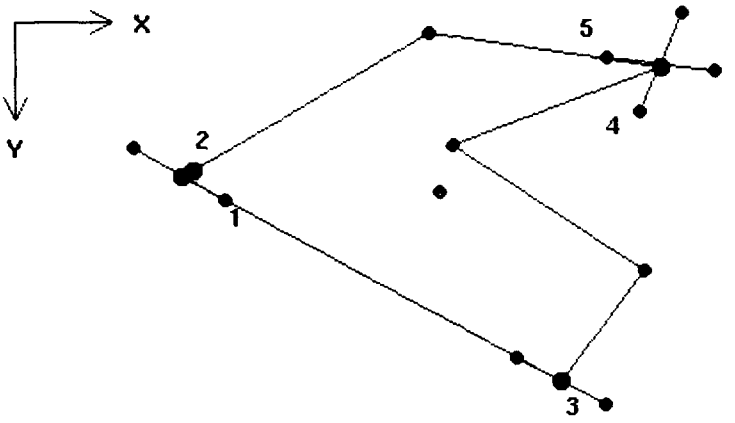


Figure 4.26: The best match of the example polygon to the polygon prototype grasp has a grasp quality measure of 0.38. In the plot, the object boundary curve of the example polygon has been superimposed on the 0.0875 quality regions. This object boundary curve has not been shifted with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.



Quality = 0.34

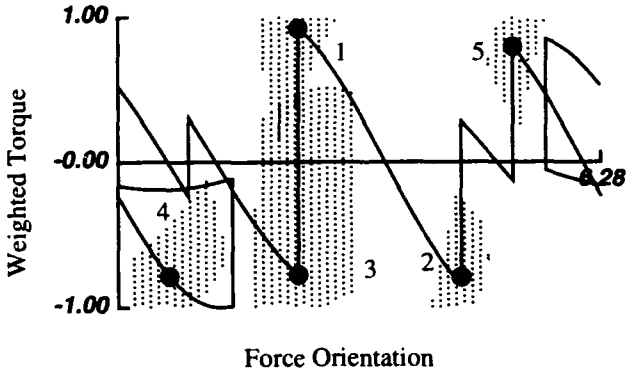


Figure 4.27: The best match of new object 1 to the polygon prototype grasp has a grasp quality measure of 0.34. In the plot, the object boundary curve of new object 1 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by 3.4 radians with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

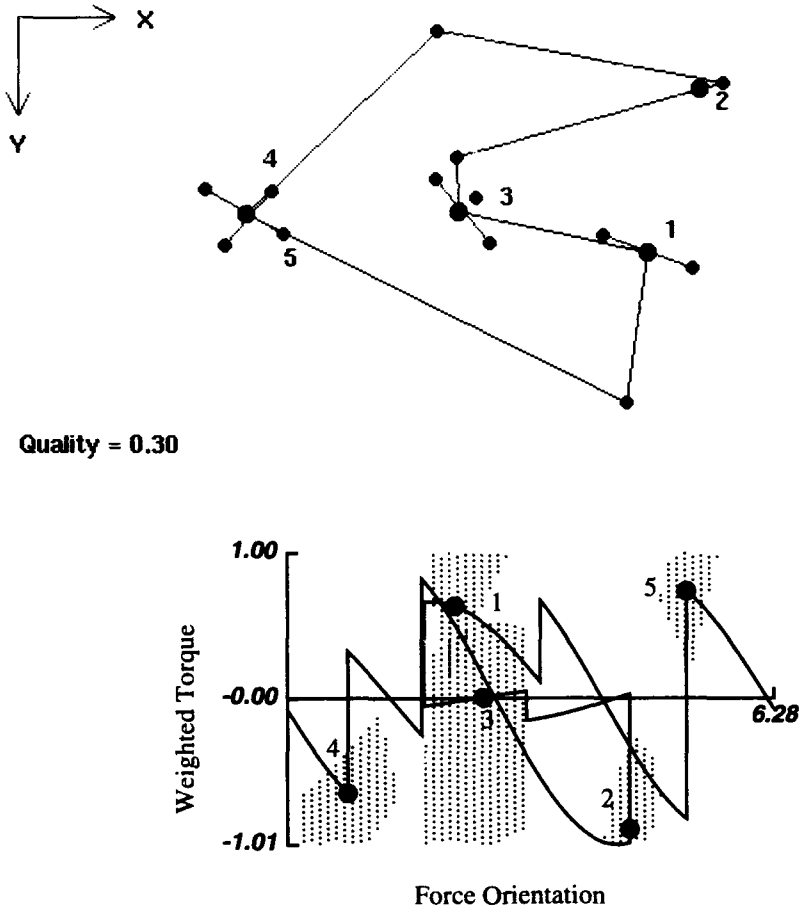


Figure 4.28: The best match of new object 2 to the polygon prototype grasp has a grasp quality measure of 0.30. In the plot, the object boundary curve of new object 2 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by -0.02 radians with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

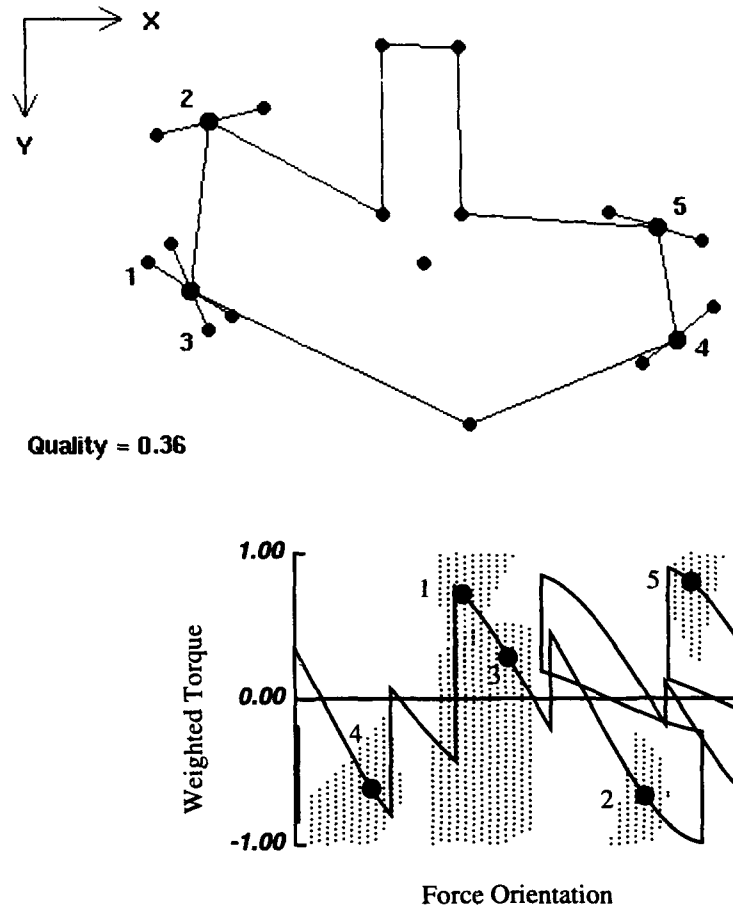


Figure 4.29: The best match of new object 3 to the polygon prototype grasp has a grasp quality measure of 0.36. In the plot, the object boundary curve of new object 3 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by 3.2 radians with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

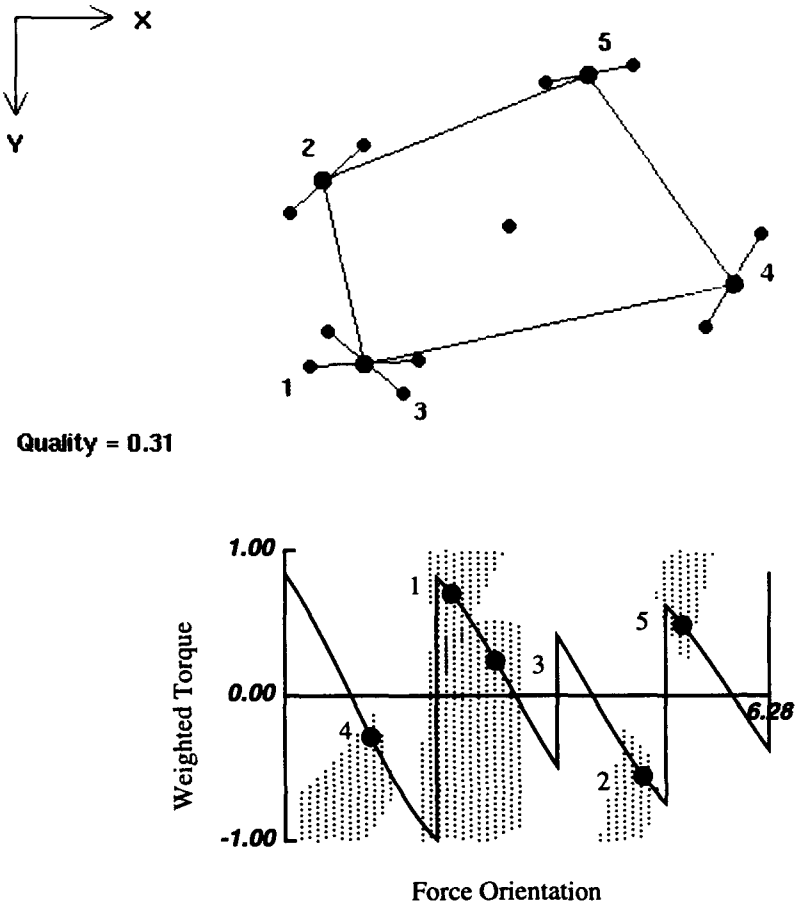


Figure 4.30: The best match of new object 4 to the polygon prototype grasp has a grasp quality measure of 0.31. In the plot, the object boundary curve of new object 4 has been superimposed on the 0.0875 quality regions. This object boundary curve has been shifted by 3.7 radians with respect to the independent contact regions of the grasp prototype. The contact wrenches of the grasp are indicated in the plot, and independent contact region sizes for the quality threshold of 0.0875 can be estimated by examining the plot.

Chapter 5

Optimizing Grasps: A Parallel Algorithm

The previous chapters considered the grasp planning problem from a limited perspective. A grasp was viewed abstractly, as a set of contact points and contact forces. This allowed the problem of constructing a high-quality grasp to be explored without considering the geometry of the world or the specifics of any particular robot hand. It was shown that good solutions to this abstract problem could be obtained by using a grasp prototype to constrain the solution space. Grasps having large independent contact regions were generated by fitting a grasp prototype to a target object. A good grasp could be achieved as long as each contact fell within its designated region.

When a more realistic problem domain is considered, a problem domain where the robot hand must achieve the given contacts, it is clear that more information is needed. An objective function derived from contact region sizes is no longer sufficient to identify a good grasp. The target object is not likely to be floating in empty space, and the robot hand is not infinitely flexible. A useful objective function must take into account the space of good hand configurations.

This chapter moves toward a more complete solution by adding two new constraints to the problem:

- the geometry and kinematics of the robot hand, and
- the geometry of obstacles in the environment.

A model of the geometry and kinematics of the hand is necessary to rule out solutions such as that shown in Figure 5.1, where there is no way for the hand to reach all three contact points at once. Knowledge of obstacles other than the target object is necessary for situations such as that shown in Figure 5.2, where the hand may be able to reach a contact point only by pushing its way through an obstacle.

Current solutions to this more complete problem are few, and those that do exist either solve problems of reduced complexity or rely exclusively on local methods. Both of these approaches have undesirable properties. Reduced complexity analyses will miss many

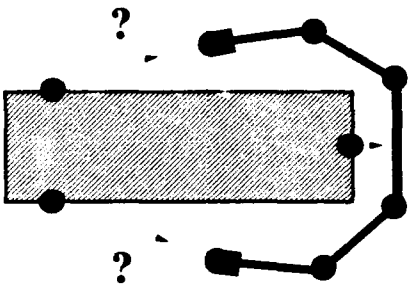


Figure 5.1: The contact points of this grasp are too far apart. The hand cannot reach all three of them.

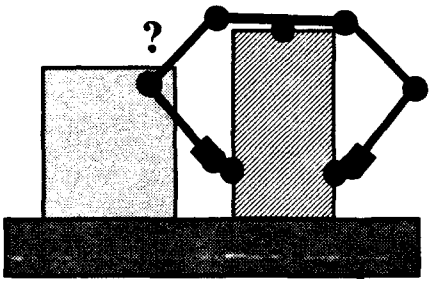


Figure 5.2: Obstacles may cause sets of contact points to be unreachable.

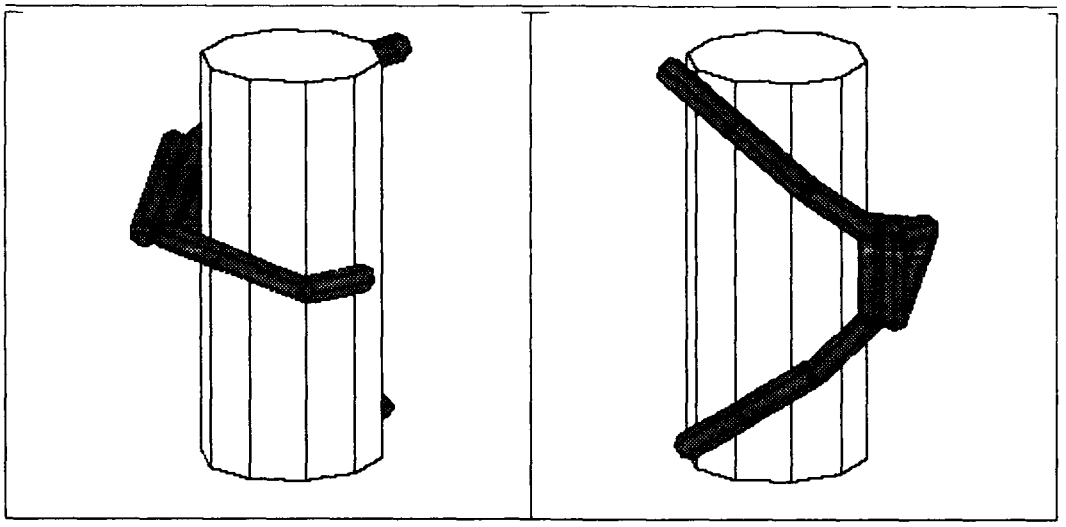


Figure 5.3: Two views of a prototype cylinder grasp.

solutions due to the rigid limits imposed on the problem. They may be unable to adapt to even modest changes in the target object or in the geometry of the environment. Local methods are able to adapt to small changes, but they provide little or no information about the structure of the solution space. This type of information is important for estimating the probability of achieving a good grasp or for comparing one grasp to another. It can be used to measure the effect of obstacles in the environment, resolve free parameters or symmetries in a grasp, or even indicate whether a solution is possible at all.

This chapter presents a parallel, dynamic programming algorithm for grasp synthesis. The algorithm is flexible, allowing for considerable variation in both hand configuration and target object geometry, and it generates a projective representation of the space of good solutions. The algorithm presented in this chapter builds directly on the results of Chapter 3. The dynamic programming advantage of this algorithm is gained by synthesizing only those grasps that match a grasp prototype as defined in that chapter.

The main contribution of this algorithm is that it satisfies a number of important grasp constraints — grasp quality, kinematic feasibility, and collision avoidance — while solving complex grasp synthesis problems with a competence that has not been previously demonstrated. Despite the complexity of the problem, useful global information about a situation is extracted, and interesting grasps are synthesized that meet the problem constraints. The algorithm does not represent a complete solution, however. Only the geometry and kinematics of the robot hand are considered. The robot arm is ignored. In addition, the algorithm finds only end grasps of a target object. There is no guarantee that any of these grasps can be reached without collisions.

Figure 5.3 shows two views of the cylindrical prototype grasp used for the examples presented in Chapter 7. Note that the example target objects are now three-dimensional.

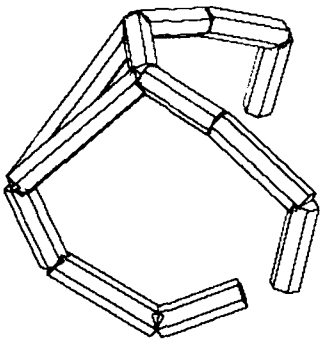


Figure 5.4: A simple model of the Salisbury hand. The hand has three fingers, each with three joints. The triangle represents the wrist.

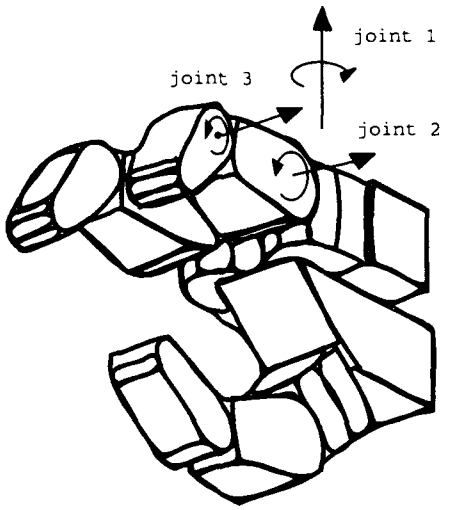


Figure 5.5: A model of the Salisbury hand, illustrating the axes of joint motion for the left finger.

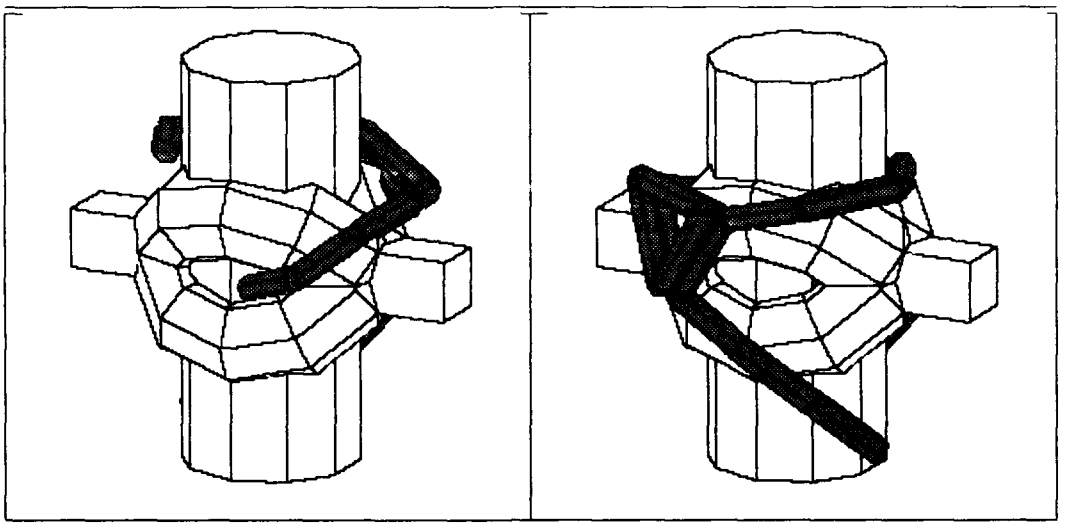


Figure 5.6: A matching grasp of a complex object that matches the prototype grasp of Figure 5.3.

The geometric model of the hand in this prototype is very simple. An isolated view of this hand model is shown in Figure 5.4. It is a skeleton model of the Salisbury hand (a better model is shown below in Figure 5.5), and it has the kinematics of that hand.

Figures 5.6 and 5.7 show the type of results demonstrated in Chapter 7. Figure 5.6 shows one grasp that has been identified for an object that is more complex than the prototype object. Figure 5.7 shows a grasp that was found to be free of collisions in a crowded environment. The second figure shows why this algorithm is only a partial solution. Although this grasp is free of collisions, it is unlikely that a robot arm would be able to maneuver the hand into this position.

The next few sections more completely define the problem solved in this chapter, outline some desired characteristics of a solution to this problem, and provide some background information. Section 5.1 gives the problem specifications, and Section 5.2 presents the notation that will be used for time complexity estimates. Section 5.3 reviews work directly related to this problem and work directly related to the solution technique of the next few chapters. Section 5.4 presents a plan for the remainder of the chapter, where a parallel algorithm is derived that incorporates hand kinematics and obstacle geometry into the grasp synthesis problem.

5.1 Problem statement

This chapter uses a definition of a grasp that includes the constraints imposed by the hand kinematics and by obstacles in the environment. It presents a technique that accommodates these new constraints while fitting a grasp prototype to a target object.

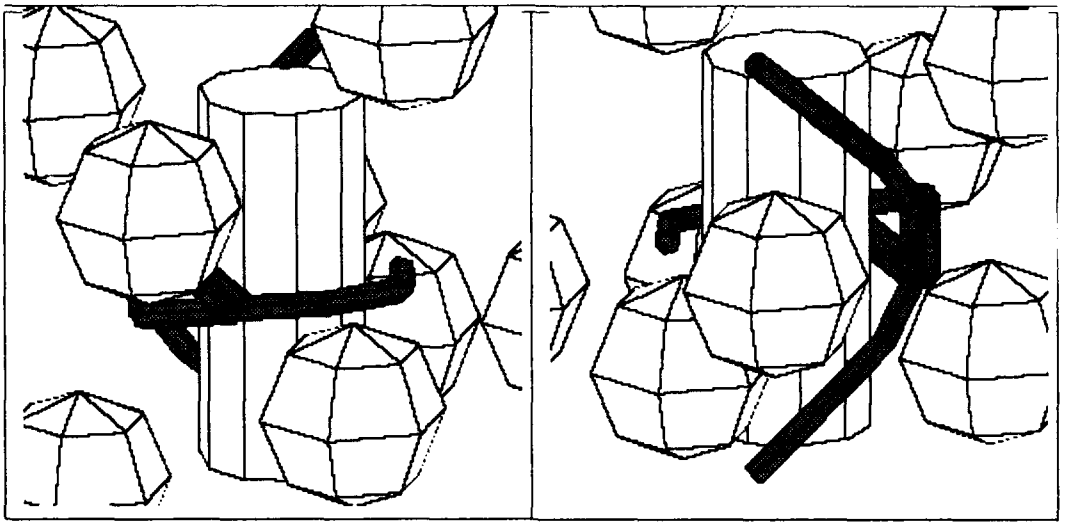


Figure 5.7: A grasp of the prototype object in a crowded environment that matches the prototype cylinder grasp of Figure 5.3.

5.1.1 Problem Specifications

The problem specifications of Chapter 3 must be modified to fully define the problem solved in this chapter. The revised specifications are shown below, with modifications shown in italics:

Definitions:

- **A grasp:** a set of contacts on a target object, *along with a kinematically feasible, collision-free hand configuration from which the contacts can be reached.*
- **A task:** defined by the task wrench space, or the space of resultant forces and torques that must be applied to the target object.
- **A grasp prototype:** an example object and a high-quality grasp of that object.

Inputs:

- A geometric model of a target object,
- *a geometric and kinematic model of a robot hand,*
- *a geometric model of the environment,*
- a grasp prototype,
- a task.

Assumptions:

- Frictionless point contacts,
- no singular contacts,
- no complex contacts,
- contact torques defined about the target object center of mass,
- grasp quality computed as in Chapter 2.

Problem:

- *Describe the space of high-quality, collision-free grasps of the target object.*

The next few paragraphs highlight the most important features of these specifications. In particular, they describe how grasp quality is computed for a proposed grasp, they outline the additional concerns encountered when adding the kinematic and geometric constraints of the robot hand, and they review the role of the grasp prototype in synthesis of high quality grasps.

Grasp Quality can be Computed from the Contacts of a Grasp

As in the previous chapter, the synthesis of any grasp requires that a set of contact points and local contact normals be found. Contact points can be placed anywhere on the target object — on a surface, an edge, or a vertex. There are no singular contacts, so contact normals can be uniquely determined by examining the local geometry of each contact. Knowing a set of contact points and the local contact normals at those points provides enough information to compute a grasp quality measure for a frictionless grasp, except in the case of complex contacts, which are discussed in Section 8.1.2.

Grasps are Now Found Using a Kinematic Model of a Hand

One major difference from the problem specification of the previous chapter is that a hand configuration must be found that achieves a high-quality set of contact points. This hand configuration must be both kinematically feasible and collision-free. If a hand configuration is kinematically feasible, the hand can physically achieve that configuration. None of the joint angles, for example, will be forced past their limits. If a hand configuration is collision-free, then the hand in that configuration does not occupy the space filled by the target object or by some obstacle in the environment. The models of the environment geometry, target object geometry, and hand geometry are used to ensure that this condition is met.

A particular robot hand is used in the examples of the next few chapters, although the various algorithms that are presented do not rely on the kinematics or geometry of this hand. The hand that is used is the Salisbury hand, which has three fingers, each with

three joints [42]. A model of this hand, showing the axes of motion of the joints of the left finger, is shown in Figure 5.5. The examples of this chapter use a skeleton model of this hand, shown in Figure 5.4. The finger and palm geometries have been simplified to form this skeleton model, but the kinematics of the hand have been preserved.

Grasp Prototypes are Used in the Solution

The next few chapters assume that a grasp prototype and a task have been provided. The grasp prototype represents a successful grasp of the example object. The grasp prototype used in the examples of Chapter 7 is shown in Figure 5.3. The task assumed in these examples is a task wrench space ball. As described in Chapter 3 (Equation 3.14), this grasp prototype and this task together provide enough information to compute contact quality measures for the contacts of any proposed grasp of a new target object. As described in Expressions 3.17 and 3.18, these contact quality measures can be used to derive objective functions for independently optimizing the placement of contacts when synthesizing a grasp. Of course, in this chapter and the chapters that follow, any contact placements that are used must also be associated with feasible hand configurations.

5.1.2 Desired Characteristics of a Solution

The algorithms presented in the sections below provide only partial solutions to the problem stated here, so some bases for evaluating these solutions are required. The list that follows describes a set of capabilities that a good solution should have. The capabilities are described briefly below, and they are reviewed when results are discussed at the end of Chapter 7:

- **Find optimal grasps.** Can we pinpoint the best grasp possibilities?
- **Evaluate how easily a grasp can be achieved.** Can we determine how easily a grasp can be achieved in the presence of model uncertainty and control error?
- **Measure the effect of obstacles in the environment.** Can we determine how the obstacles have affected the space of good grasps?
- **Measure the effect of the target object geometry.** Can we determine how the difference between the geometry of the target object and the geometry of the grasp prototype has affected the space of good grasps?
- **Select good values for parameters of symmetry in a grasp.** Can we exploit the fact that variations in the geometry of the target object and variations in the geometry of the environment break up the symmetries in a grasp and allow us to select good values for symmetry parameters?

5.2 Notation

Estimates of time complexity are sprinkled throughout this chapter and the next. These estimates require the use of the following terms:

L	=	total number of links in the robot hand
F	=	total number of fingers in the robot hand
f_E	=	number of features (e.g. face, edges, vertices) in the environment
f_T	=	number of features in the target object
f_L	=	number of features in a link of the robot hand
P	=	number of processors available
R^3	=	number of position space samples (position space is a 3D cube)
S	=	length of one side of the position space cube
O^3	=	number of orientation space samples (orientation space is 3D)
d_{angle}	=	size of a joint angle range
d_{link}	=	length of a link, joint to joint, along the link axis

The expressions used to calculate contact quality and grasp quality values in the next few chapters are more complex than the expressions used in Chapter 3. In addition, these expressions are subject to a greater variety of constraints such as joint limits and collisions. For this reason, pseudocode algorithms rather than simple expressions are used throughout the rest of the report (e.g. Algorithm 5.1 in Section 5.6). Although the lower level functions used in these algorithms are not described in the text of these chapters, they are expanded in the complete algorithm listing of Appendix A. This appendix shows, for example, that function **hand_configs_of(hand_model)** (reference point C in Algorithm 5.1) returns a set of configurations of the modelled robot hand, sampled over the kinematically feasible space of possibilities.

5.3 Previous Work

Solutions to the more complete problem addressed in this chapter have been demonstrated for grasping using a parallel jaw gripper. Lozano-Pérez et al. [30], Pertin-Troccaz [38] and others [26] [22] [36] construct grasps for varying target object geometries and in environments containing obstacles. Although they use heuristics to increase search speed, much of the search space can be analyzed in a reasonable amount of time on a parallel computer (see [29]).

Previous work on standard grasping behaviors for articulated hands allows for some variation in target object geometry. Bard and Troccaz [3] handle varying target object geometries by decomposing each target object into ellipsoids. A standard strategy for grasping ellipsoids is tried at promising areas on the target object. Nguyen and Stephanou [34]

parameterize standard grasps along several dimensions. Arbib et al. [1] describe an architecture for controlling ballistic motions of a robot hand into a grasp that can incorporate some high-level visual or tactile feedback.

Brock [5] shows that grasping strategies for articulated hands can be made very robust to uncertainty in the configuration of a known target object. His system also allows for a limited amount of obstacle avoidance.

Local control can be used to improve a precomputed initial grasp. Jameson [20] uses local control to improve grasp quality, and Pollard [40] uses local control to eliminate collisions from an initial grasp.

In general, solutions that consider problems caused by non-standard object geometry or by the presence of obstacles in the environment either apply to reduced degree-of-freedom problems, such as grasping with a parallel jaw gripper, use abstract descriptions of target object geometry (e.g. by decomposing the target object into ellipsoids), or use local control to improve a precomputed starting grasp. The solution presented in this chapter represents a different approach, one that produces a global description of the range of grasps that match a given grasp prototype. This global information can give us an idea of the shape of the solution space, or tell us whether any solutions are possible at all. The algorithm presented in this chapter is unique in that it allows for substantial flexibility in hand configuration while extracting this kind of global information.

5.4 Chapter Overview

The goal of this chapter is to describe the space of high quality, collision-free grasps of a target object in a given environment. The remaining sections of this chapter present an algorithm designed to meet this goal. This algorithm builds on the work of Chapter 3, using a generalized grasp prototype to localize contact quality computations to the individual contacts of a grasp. The challenge of this chapter is to add the constraints of hand kinematics, hand geometry, and environment geometry to the grasp synreport problem.

Section 5.5 begins by presenting a brute force approach to synthesizing and searching a space of grasp quality measures. Section 5.6 shows how the grasp prototype can be used to avoid repeating the expensive grasp quality computation over the solution space. Section 5.7 shows how this approach can be greatly improved to reduce search space complexity, and Section 5.8 describes how parallelism can be added to increase processing speed. Section 5.9 concludes with a summary of the chapter.

5.5 Examining all Hand Configurations

This section describes a brute force approach to synthesizing high quality, kinematically feasible, collision-free grasps of a target object. This approach does not use the prototype to guide the search, but it shows how the most general version of the problem can be formulated. The paragraphs below first describe the search space of the problem and then show how optimal grasps can be extracted from this space.

The grasp synthesis problem of this chapter can be stated as the problem of finding a hand configuration that represents a good grasp. To specify a hand configuration requires selecting a position and orientation of the wrist and a position for each joint angle of the hand. For a hand having j rotary joints, the search space is described by:

$$\Re^3 \times SO(3) \times S^1 \times \cdots \times S^1 \quad (S^1 \; j \text{ times}),$$

where $\Re^3$ are three parameters specifying the position of the hand, $SO(3)$ specifies a three-dimensional orientation of the hand, and $S^1 \times \cdots \times S^1$ specifies any set of j joint angles for the joints of the hand. The hand used in the examples of this report has 9 joints, and so 15 parameters must be provided to completely specify a hand configuration.

For any hand configuration, a grasp quality measure can be computed. The first step in computing the grasp quality measure is to exclude those portions of the space of hand configurations that do not meet one of the following requirements for a grasp:

1. each of the joint angles must fall within the kinematic limits of that joint,

2. the hand in the given configuration must be free of collisions,

3. the hand in the given configuration must be in contact with the target object.

The second step in computing a grasp quality measure over this space of hand configurations is to collect the contact wrenches resulting from unit applied force at the contacts between the hand and target object. From these contact wrenches and the given task, here a wrench space ball, grasp quality can be computed as stated in Equation 2.25. This grasp quality measure provides an estimate of the ability of the grasp to resist unknown disturbance forces. It requires constructing a wrench space convex hull from the contact wrenches of the grasp and measuring the size of the largest wrench space ball that fits within this convex hull.

The best grasp could be found, as described in Chapter 3, by optimizing grasp quality over the space of hand configurations or by finding a large region of hand configurations that represents a continuous space of good grasps. The former approach will find the grasp best suited for the given task, but the latter approach will result in grasps that are robust to errors in sensing the precise location of the target object or errors in controlling the motion of the hand. Either approach will in general require examining large portions of the search space. Any naive implementation of this algorithm will be able to easily exclude only the most obviously impossible solutions. For example, hand configurations with an illegal set of joint angles, or hand configurations too far from the target object to achieve contact with that object can easily be eliminated from consideration. In more complex situations, however, it becomes more difficult to eliminate poor solutions, and the remaining search space will be large. The sections below show how a grasp prototype can be used to avoid computing the grasp quality measure at each point in this space. They also show how a grasp prototype can be used to drastically reduce the size of the search space that must be examined.

5.6 Prototypes can Reduce Quality Computation

The grasp prototype can be used as described in Chapter 3 to avoid computing grasp quality for any hypothesized set of contacts between the robot hand and the target object. This section presents a simple technique for combining the new constraints of hand kinematics and obstacles into this use of a grasp prototype. The technique presented in this section is not practical, but it does show how the grasp prototype can be used with these new constraints, and it helps to motivate the algorithms presented in the sections that follow.

Chapter 3 showed that the expensive operation of computing a grasp quality measure can be avoided by using a grasp prototype. This is possible because the prototype allows contact quality measures to be computed for each contact of a grasp independently. A lower bound can be placed on the grasp quality measure of a hypothesized grasp by simply taking the minimum contact quality measure over the contacts of the grasp. As long as a contact wrench with a contact quality measure above some threshold can be found to match each contact of the grasp prototype, the grasp quality measure of that grasp is guaranteed to be above the given threshold.

Algorithm 5.1 shows an algorithm that could be used for computing lower bound grasp quality measures for grasps that are kinematically feasible and collision-free. The function **lower_bound_gq_simple** requires as inputs a grasp prototype, a model of the hand, a model of the target object at the appropriate place in the workspace, and models of a set of obstacles also at appropriate points in the workspace. The function returns an array of grasp quality measures computed over the search space of the problem. Although many of the lower level functions used by this algorithm are not described in the text of this chapter, they are expanded in the complete algorithm listing of Appendix A.

The innermost loop of Algorithm 5.1 (reference point C) samples the space of hand configurations. In the previous section, three conditions were stated for a hand configuration to be considered a possible grasp:

1. each joint angle must fall within the kinematic limits of that joint,
2. the hand in that configuration must be free of collisions,
3. the hand in that configuration must be in contact with the target object.

Checks for these conditions can be found in the inner loop of Algorithm 5.1. It is assumed that **hand_configs_of** returns only kinematically feasible hand configurations. Function **no_hand_collisions?** (D) ensures that each hand configuration considered further is free of collisions between the hand and the **object** or between the hand and any of the **obstacles**. Finally, **link_contact?** (E) ensures that some contact is made between the hand and the **object**.

If the three conditions for a possible grasp are met, then the lower bound grasp quality measure that is stored in array **gq** is computed exactly as it was in Chapter 3. Contact quality measures (G) are computed for each contact of the grasp. The minimum of these contact quality measures is taken to be the lower bound grasp quality measure for the

grasp (H). Computing this minimum is more efficient than constructing and testing the convex hull of the contact wrenches of the grasp, and this increase in efficiency is one part of the appeal of using a grasp prototype.

Unfortunately, in addition to this inner loop over hand configurations, which was present in the brute force algorithm of the Section 5.5, Algorithm 5.1 has two outer loops. These loops are required to match a target object to the grasp prototype and to establish an assignment of links of the hand to contacts of the grasp prototype. The outermost loop (A) specifies an alignment of the grasp prototype to the target object. This loop was also present in Expressions 3.17 and 3.18 of Section 3.6 above. A relative orientation of the grasp prototype with respect to the target object is necessary to fix the coordinate system in which all contact quality measures are computed. This coordinate system must be fixed because the contact quality measures depend on knowing approximate values for the other contact wrenches of the grasp.

The second loop (B) specifies a **contact_assignment**, an assignment of links of the hand to contacts of the grasp. In order to evaluate contact quality, each contact of a grasp must be assigned to a contact of the grasp prototype. When the kinematics of the robot hand are considered, a contact must also be associated with some link of the robot hand (in these algorithms, the palm of the hand is also described as a link). Specifying a **contact_assignment** provides this association. It ensures that a grasp contains a complete set of contacts by specifying the link on which each contact should be found. The grasp quality estimate then has a positive value only when a contact with a positive contact quality measure is found for each link-contact assignment specified.

Notice that Algorithm 5.1 not only assumes that the assignment of each contact to a link of the hand has been specified, but it also assumes that a single representative contact wrench can be extracted for any link configuration in which the link contacts the target object (F). This assumption restricts the contact between the link and the target object to be approximated as a *simple contact*. It is possible that a contact between a link and the target object could be best described using a space of contact wrenches rather than a single contact wrench (e.g. if there is friction, if the link lies flat on a face of the target object, or if the fingertip is placed in a concavity). This type of *complex contact* is discussed in Section 8.1.2 below.

The synthesis of optimal grasps of a target object would in general require searching through the grasp quality space **gq** computed in Algorithm 5.1. This space could be searched to find a grasp with an optimal grasp quality estimate, or to find large regions of grasps having a grasp quality estimate above a certain threshold. With either approach it is difficult to choose promising areas of the solution space from which to initiate a search, and so the size of this space is important.

The use of a grasp prototype has made the process of estimating grasp quality for any hypothesized grasp more efficient, but the two outer loops of Algorithm 5.1 show that it has also expanded the size of the complete grasp quality space. In this simple algorithm, there is little advantage to using the grasp prototype. In fact, this algorithm has two major problems. One problem is that the solution space is too large for any significant portion of this space to be constructed. The other is that similar information must be

```
   {compute entire space of lower bound gq measures based on cq measures}
   lower_bound_gq_simple (prototype, hand_model, object, obstacles)
   begin
      {align grasp prototype with target object}
A     for orient in workspace_orients() do
         {assign links of the hand to contacts of the prototype}
B        for contact_assignment in contact_assignments_of(hand_model, prototype) do
            {each hand config is a possible grasp}
C           for hand_config in hand_configs_of(hand_model) do
            begin
               gq[orient, contact_assignment, hand_config] := MIN_QUALITY;
D              if no_hand_collisions?(hand_config, hand_model, object, obstacles) then
               begin
                  {lower bound gq is minimum cq over contacts}
                  min_cq := MAX_QUALITY;
                  for contact in contacts_of(prototype) do
                  begin
                     link := link_of(contact, contact_assignment);
                     link_config := link_config_of(link, hand_config);
                     cq := MIN_QUALITY;
E                    if link_contact?(link, link_config, hand_model, object) then
                     begin
F                       wrench := contact_wrench(link, link_config, hand_model, object);
G                       cq := contact_quality(orient, contact, wrench, prototype);
                     end
                     if cq < min_cq then
                        min_cq := cq;
                  end
H                 gq[orient, contact_assignment, hand_config] := min_cq;
               end
            end
      return(gq);
   end
```

Algorithm 5.1:

recomputed many times. The sections below expand upon these problems and show how they can be overcome to make more effective use of the grasp prototype.

5.7 Prototypes can Reduce Search Complexity

The previous section described a simple algorithm for incorporating hand kinematics, hand geometry, and environment geometry into the grasp synthesis problem. This simple algorithm had two major problems:

1. the search space specified was very large, and

2. similar partial solutions were recomputed many times.

The first problem is that lower bound grasp quality measures must be computed for all hand configurations, for all contact assignments, and for all orientations of the grasp prototype with respect to the target object. The innermost and outermost of these loops are very expensive. For the hand used in this report, 15 parameters are required to specify a hand configuration, and 3 parameters are required to specify the orientation of a grasp prototype with respect to a target object. This results in an 18 dimensional parameter space over which to compute an estimate of the grasp quality measure. Given this dimensionality, even the solution— the space of high quality grasps—will in general be too large to write down.

The second problem is that very similar partial solutions are recomputed many times. For any given value of parameters ***orient*** and **contact_assignment** in Algorithm 5.1, contact quality information depends on the link of the hand and the configuration of that link (F and G). The grasp quality estimate, however, is computed by looping over the space of hand configurations (C). The same link, at a similar link configuration may appear many times in this space of hand configurations, and so similar contact quality estimates are computed many times in this simple algorithm.

This section shows how both the problem of a large solution space and the problem of repeated derivation of partial solutions can be overcome. To reduce the size of the solution space that must be constructed, the algorithm presented in this section computes a projection of the complete solution space. The grasp prototype makes this projection more useful, because it constrains the possible interpretations of the projection. To enable partial solutions to be reused, this algorithm uses the grasp prototype to compute and store an array of contact quality measures over all configurations of all links of the hand. It then chains the best link configurations together in kinematically feasible ways to form complete high quality, collision-free grasps.

Section 5.7.1 begins with an overview, describing the top level functions of the algorithm presented in this section. Section 5.7.2 covers the subproblem of computing the contact quality spaces for each contact of a grasp, and Section 5.7.3 shows how the grasp quality space can be constructed by combining results from the precomputed contact quality spaces. Section 5.7.4 presents a summary.

5.7.1 Algorithm Overview

The problems of the simple algorithm of the previous section are resolved in the paragraphs below by reducing the space of solutions that must be stored, and by allowing partial solutions to be reused. This section examines these goals in more detail and presents a top level function for computing the grasp quality space in a way that achieves these goals.

Reducing the Size of the Solution Space

One of the problems with Algorithm 5.1 is that even after parameters **orient** and **contact_assignment** are specified to align the grasp prototype with the target object and to assign contacts of the grasp to links of the hand, the algorithm requires computing a grasp quality measure for every hand configuration. This results in an extremely large grasp quality space. The algorithm presented in this section also requires that parameters **orient** and **contact_assignment** be specified, but rather than computing a grasp quality measure over the entire space of hand configurations, it computes a grasp quality measure over the much smaller space of *wrist configurations.*

Since any wrist configuration can support a large space of hand configurations, and thus a large space of possible grasps, it is necessary to specify how a single grasp quality measure will be attached to a wrist configuration. If we wish to find optimal grasps based on the grasp quality measure, it is useful to store the *best* grasp quality measure that can be achieved from each wrist configuration. A complete hand configuration associated with this best grasp quality measure can also be stored for fast retrieval.

Once all wrist configurations have been associated with optimal grasp quality measures, regions of wrist configuration space can be identified that are associated with grasp quality measures above a given threshold. The identification of these regions provides a substantial amount of information about the shape of the solution space. It indicates whether any good solutions exist at all, it provides pointers to the best of these solutions, and it shows how wrist configurations are blocked or unblocked by the addition or removal of obstacles in the environment.

Figure 5.8 shows how a representation of the space of good wrist configurations might be used. The shaded areas in the figures represent wrist positions from which some good grasp can be found for the two-dimensional robot hand in the figure. These regions do not represent the complete space of good grasps, because at any one wrist position a family of grasps may be possible. The regions do, however, provide useful information about the structure of the solution space. The shapes of the regions in these examples are different, and this reflects the difference in the structure of the environment.

A representation of optimal grasp quality measures computed over the space of wrist configurations might not seem to be sufficient, because it captures only one slice of the complete solution space. The examples in Chapter 7 show, however, that this space does provide useful global information. In addition, it serves as a good starting point for a more complete search. Partial solutions constructed while computing this space allow an efficient guided search to be performed through the entire space of legal solutions. Section 5.9 discusses how this is done.

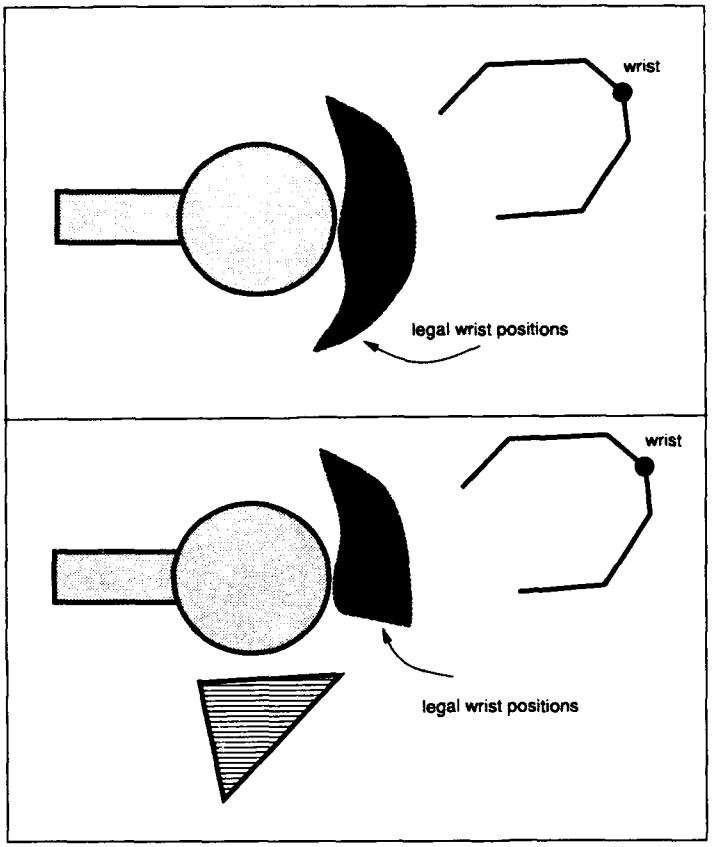


Figure 5.8: Regions of good wrist configurations indicate where solutions are possible and how obstacles affect the range of possible solutions. This figure shows how the good space of wrist positions for a two-dimensional robot hand is constrained by the presence of the target object only (top) and by the target object with an obstacle (bottom).

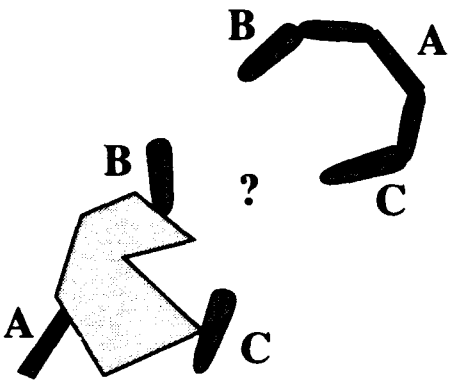


Figure 5.9: Relative link configurations are tightly constrained by the kinematics of the robot hand. The contact configurations shown here do not even come close to matching the kinematics of the hand.

Reusing Partial Solutions

The previous paragraphs outlined how the solution space of Algorithm 5.1 might be reduced to a space that can be computed. Algorithm 5.1 also had a second problem that made it inefficient: in this algorithm very similar or identical partial solutions were recomputed many times rather than being stored and reused. In this algorithm, full advantage is not taken of the ability of a grasp prototype to allow contact quality measures to be computed for each contact independently.

This section proposes a better solution. Contact quality spaces are computed for each link of the hand, for each configuration of that link. Complete solutions must be formed by selecting a high quality link configuration associated with each contact of a grasp. The selection of a good set of link configurations is constrained, however, by the kinematics of the hand (Figure 5.9). The sections below show how high-quality link configurations are computed and chained together to form complete high-quality grasps.

The Top Level Function

The top level function for the algorithm presented in this section is shown in Algorithm 5.2. Function **lower_bound_gq_1** requires the same inputs as Algorithm 5.1: a grasp prototype, a model of the hand kinematics and geometry, and models of the target object and environment.

The first goal of this new algorithm is to compute a projection of the complete solution space. In Algorithm 5.2 (reference point D) the grasp quality array **gq** is indexed by parameters **orient** and **contact_assignment** as in the simple function of Algorithm 5.1. Function **compute_gq_1** of Algorithm 5.5 (C), shows that **gq** is further indexed by the six-dimensional wrist configuration parameter **wrist_config** rather than the fifteen-dimensional

```
   lower_bound_gq_1(prototype, hand_model, object, obstacles)
   begin
      {align grasp prototype with target object}
A     for orient in workspace_orients() do
         {assign links of the hand to contacts of the prototype}
B        for contact_assignment in contact_assignments_of(hand_model, prototype) do
         begin
            {compute contact quality array cq }
C           cq := compute_cq_1
                    (orient, contact_assignment, prototype, hand_model, object, obstacles);
            {compute grasp quality array gq}
D           gq[orient, contact_assignment] := compute_gq_1 (cq, hand_model);
         end
      return(gq);
   end
```

Algorithm 5.2:

hand configuration parameter **hand_config** of the simple Algorithm 5.1 (H). The grasp quality measure has been projected from the original fifteen-dimensional space of hand configurations onto the six-dimensional space of wrist configurations. Section 5.7.3 below describes how this projection is performed.

The second goal of this new algorithm is to store and reuse partial solutions. In the top level function of Algorithm 5.2, a contact quality array **cq** is computed first (C) and then supplied to a second function that puts these contact quality measures together to form the grasp quality array **gq** (D). Because contact quality measures must be computed for all links of the hand and all configurations of these links, the array of contact quality measures is indexed by a **link** and a **config** (Algorithm 5.3, reference point A).

In order to compute each space of contact quality measures, two parameters must be specified, and these parameters are represented in the outer loops of the function. The first of these parameters, **orient** (A), is the alignment between the grasp prototype and the target object. This orientation parameter is needed, as in Chapter 3, because contact quality measures must be evaluated with respect to the same coordinate system. If this coordinate system is not specified, then a lower bound grasp quality measure cannot be computed by examining each contact separately. Without a common coordinate system, there is no guarantee that the other contacts of the grasp will play the roles that are expected of them.

The second parameter **contact_assignment** (B) is also needed for computing the contact quality measure **cq**. This parameter indicates which links of the hand are assigned to which

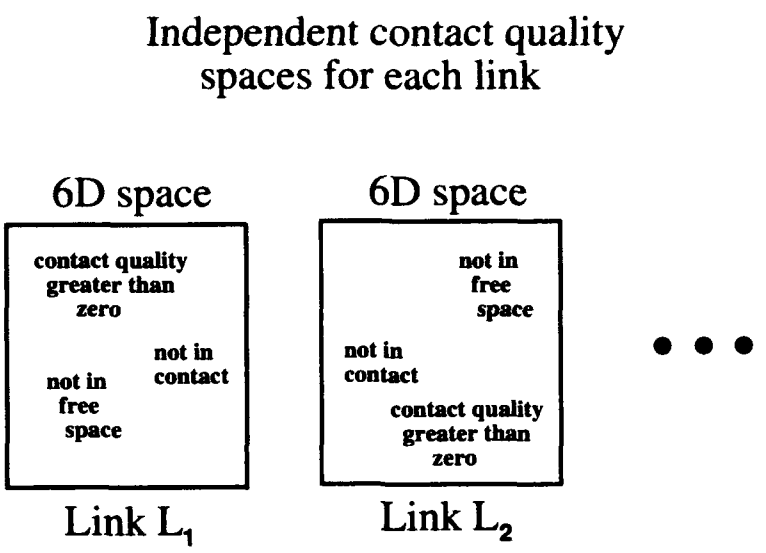


Figure 5.10: The contact quality computation generates a six-dimensional space of contact quality measures for each link of the robot hand. Some regions will have negative contact quality measures due to collisions between the link and some object in the environment, or due to the absence of a contact between the link and the target object when such a contact is expected.

contacts of the grasp. Explicitly specifying this assignment is the cleanest and easiest way to ensure that every grasp with a positive grasp quality measure will have a contact with a positive contact quality measure to match every contact of the grasp prototype.

5.7.2 Computing Contact Quality Space

Top level function **lower_bound_gq_1** of Algorithm 5.2 begins by computing a space of contact quality measures **cq** and then uses these results to construct the complete grasp quality space **gq**. Computing and storing a space of contact quality measures allows these measures to be reused. This is important because a given link in a given configuration may form part of many different grasps.

This section shows how the contact quality space is computed. The top level function **compute_cq_1** is shown in Algorithm 5.3. When this function is called, parameters **orient** and ***contact_assignment*** have been specified, and so a contact quality measure can be computed over all configurations of all links of the hand. Algorithm 5.3 simply displays the loops over these parameters. The result of executing function **compute_cq_1** is a set of spaces of contact quality measures, computed over all configurations of all links of the hand (Figure 5.10).

```
compute_cq_1 (orient, contact_assignment, prototype, hand_model, object, obstacles)
begin
   {unwind contact quality calculation for all links and link configurations}
   for link in links_of(hand_model) do
      for config in workspace_configs() do
A        cq[link, config] :=
            compute_link_cq_1
               (link, config, orient, contact_assignment, prototype, hand_model,
                  object, obstacles);
   return(cq);
end
```

Algorithm 5.3:

Quality of a Link Configuration

For a contact quality measure to be meaningful, a given link in a given configuration must be free of collisions and in contact with the target object. Then a contact wrench can be extracted from the link-target object contact and the contact quality measure can be computed for that contact wrench. Function **compute_link_cq_1** of Algorithm 5.4 outlines this process. The function begins by checking that the link lies in free space (A). If the link has not been assigned to any particular contact, this condition is sufficient for that link to be capable of forming a part of any good grasp (B). Otherwise, if the link has been assigned to some contact, the algorithm checks that a contact exists (C), finds a representative contact wrench (D), and computes the contact quality measure for this contact wrench (E). If a link in the given configuration does not meet any one of the constraints necessary for this link to form part of a good grasp, then a negative contact quality value is returned. The contact quality measure for a given link and configuration can be interpreted as follows:

MIN_QUALITY	$\Longrightarrow$	collision
MAX_QUALITY	$\Longrightarrow$	no collision, no contact assignment
0	$\Longrightarrow$	no collision, contact assignment, no contact
other	$\Longrightarrow$	no collision, contact assignment, contact

Time

The time required to compute the contact quality array varies with the size of the array and the complexity of a single contact quality computation. Function **compute_cq_1** of Algorithm 5.3 indicates that the contact quality array is indexed by a **link** and a **config**

```
   compute_link_cq_1
      (link, config, orient, contact_assignment, prototype, hand_model, object, obstacles)
   begin
      cq := MIN_QUALITY;
      {link cannot be part of a good grasp if there is a collision}
A     if no_collisions_1?(link, config, hand_model, object, obstacles) then
         {if link has no contact assignment, it is sufficient there be no collisions}
         if no_contact_assignment?(link, contact_assignment) then
B           cq := MAX_QUALITY;
         else
            {if link has a contact assignment, it must make contact}
C           if no_link_contact?(link, config, hand_model, object) then
               cq := 0;
            else
            begin
               {now we can compute contact quality}
               contact := contact_of(link, contact_assignment);
D              wrench := contact_wrench(link, config, hand_model, object);
E              cq := contact_quality(orient, contact, wrench, prototype);
            end
      return(cq);
   end
```

Algorithm 5.4:

(A). For every **link**, **config** pair, the link is checked for collisions with the target object or with some obstacle in the environment. This requires time dependent on the complexity of the geometric description of the environment and the complexity of the geometric description of the link. Time to execute function **compute cq_1** of Algorithm 5.3 can be estimated as follows (for notation see Section 5.2):

$$O(L\,(f_E f_L + f_T f_L)), \tag{5.1}$$

sampled over

$$\Re^3 \times SO(3), \tag{5.2}$$

which is the complete six-dimensional space of configurations of a three-dimensional body, in this case the link.

5.7.3 Estimating Grasp Quality Space

The previous sections have shown how a contact quality array **cq** can be computed for all configurations of all links of the hand. Somehow these links must be put together in kinematically feasible ways to form complete grasps. This section outlines how this is done.

Algorithm 5.5 shows function **compute_gq_1**, used to compute a space of grasp quality measures. *This function requires as inputs only* the contact quality array **cq** and the parameter **hand_model**, which provides information describing how the links of the hand must be connected. The return value of this function is grasp quality array **gq**, computed over the space of wrist configurations (C). The space of wrist configurations is a projection of the more complete solution space of hand configurations, and the goal is to optimize the grasp quality estimate when performing this projection. This means that the optimal grasp quality estimate must be computed and stored for all hand configurations associated with any given wrist configuration.

Function **compute_gq_1** is divided into two parts: first, the best links are strung together to form the optimal quality fingers associated with each wrist configuration (A), and then a simple loop is executed to compute the grasp quality measure for each wrist configuration from the finger quality measures associated with that wrist configuration (B). Recall that a grasp quality measure is estimated by taking the minimum of the contact quality measures over the contacts of a grasp. As we will see, each finger quality measure is also computed in this way; the finger quality is estimated by taking the minimum of the contact quality measures over the contacts of a finger. The appropriate grasp quality measure for each wrist configuration is then computed by taking the minimum of the finger quality measures over the fingers of the hand.

Quality of a Finger

Grasp quality computation in function **compute_gq_1** begins with the computation of optimal finger quality measures. This section shows how these finger quality measures are determined. For any given wrist configuration, we would like to find the best quality

```
   compute_gq_1 (cq, hand_model)
   begin
      {compute finger quality arrays fq for all fingers}
A     for finger in fingers_of(hand_model) do
      begin
         prox_link := prox_link_of(finger, hand_model);
         fq[finger] := finger_quality_1(prox_link, cq, hand_model);
      end

      {compute grasp qualities gq for all wrist configurations}
B     for wrist_config in workspace_configs() do
      begin
         {minimize grasp quality gq over finger qualities}
         gq[wrist_config] := MAX_QUALITY;
         for finger in fingers_of(hand_model) do
         begin
            prox_config := prox_link_config_of(finger, hand_model, wrist_config);
            if fq[finger, prox_config] < gq[wrist_config] then
C              gq[wrist_config] := fq[finger, prox_config];
         end
      end
      return(gq);
   end
```

Algorithm 5.5:

measure of a finger that results from some configuration of that finger based at the given wrist configuration.

Algorithm 5.6 describes recursive function **finger_quality_1**, which is designed to compute an array of finger quality measures. The function assumes that a finger is a serial chain of links. It is first called with the proximal link of the given finger, the link closest to the wrist, and it terminates when the distal link, the link containing the fingertip, is reached (A). Each call to the recursive function works as follows:

1. The function is invoked for a particular **link**, with a **cq** array and a **hand_model**.

2. Finger quality values are computed for the distal neighbor of that link, **next_link**, to obtain the array **subchain_quality** (B), which can be indexed by a configuration of link **next_link**.

3. For each possible **config** of the input **link** (C), the qualities of the subchains that can be reached from that link are examined. These subchains are compared by sampling the distal joint angle of the input **link**, **theta** (D). From **link**, **config**, **theta**, and the **hand_model**, a unique configuration of **next_link**, the distal neighbor of **link**, can be found (E). This configuration, **next_config**, is used as an index into the **subchain_quality** array (F), which has already been computed in step 2.

4. To maximize the quality of the finger under construction, the quality of the best subchain reachable from a **link** and **config** over all joint angles **theta** is retained (G). This quality value is stored in variable **max_sq**.

5. To minimize the finger quality measures over the contacts of each finger for each **config** of **link**, the minimum of **max_sq** and the contact quality measure corresponding to that **link** and **config** is retained in **fq[config]** (H).

6. The array of subchain qualities is returned for the subchain from the distal link of the finger up through **link** for the entire space of configurations of **link**. Each subchain quality measure results from a subchain that maximizes the minimum of the contact quality measures over the contacts of the subchain.

Figure 5.11 unwinds part of this recursive process, showing the path along which an optimal solution for one finger at one proximal link configuration is retrieved.

Time

The paragraphs above have described an algorithm which uses an array of contact quality measures for each configuration of each link of the hand to compute an array of grasp quality measures over the space of wrist configurations. The function described is function **compute_gq_1** of Algorithm 5.5. The most expensive part of this function takes place in the recursive calls to function **finger_quality_1** of Algorithm 5.6. This function is eventually called once for each link of the hand. Adding a link to a current array of subchain quality

```
   finger_quality_1(link, cq, hand_model)
   begin
      {terminate at distal link}
A     if no_next_link?(link, hand_model) then
         return(cq[link]);

      {get subchain quality for next link}
      next_link := next_link_of(link, hand_model);
B     subchain_quality := finger_quality_1(next_link, cq, hand_model);

      {get subchain quality for current link}
C     for config in workspace_configs() do
      begin
         {maximize next link subchain quality over joint angles}
         max_sq := MIN_QUALITY;
D        for theta in joint_angles_of(link, hand_model) do
         begin
E           next_config := next_link_config_of(theta, link, hand_model, config);
F           if subchain_quality[next_config] > max_sq then
G              max_sq := subchain_quality[next_config];
         end

         {take minimum of link and current subchain quality values}
H        fq[config] := min(cq[link, config], max_sq);
      end
      return(fq);
   end
```

Algorithm 5.6:

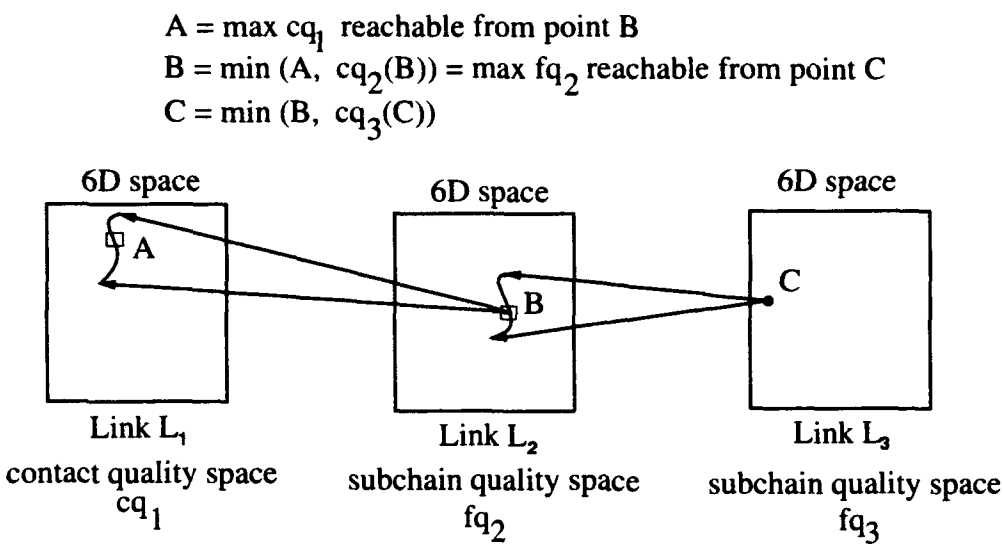


Figure 5.11: The link contact quality spaces can be chained together recursively to produce optimal subchain quality spaces of equal size.

measures requires scanning a range of joint angles (D), and the array is filled for all link configurations (C). This gives us a first order time estimate of:

$$O(L) \tag{5.3}$$

sampled over

$$\Re^3 \times SO(3) \times S^1, \tag{5.4}$$

where $\Re^3 \times SO(3)$ represents the sampling over link configurations, and S^1 represents the sampling over range of joint angles.

5.7.4 Summary

This section has addressed two shortcomings observed in the simple algorithm of Section 5.6. First, the size of the solution space has been reduced. Grasp quality measures are computed over the space of wrist configurations rather than over the much larger space of hand configurations. Second, partial solutions are precomputed so that they can be reused. Contact quality measures are constructed for each configuration of each link of the hand, and then the best quality link configurations are chained together in kinematically feasible ways to produce the solution space of grasp quality estimates. Unfortunately, this algorithm is still very computationally intensive. The next section, and the entirety of Chapter 6 are devoted to making this algorithm practical to execute on today's computers.

5.8 Parallelism can Increase Processing Speed

The previous section described an algorithm that used a grasp prototype to compute a projective space of grasp quality measures. The grasp quality measures were computed over the space of wrist configurations, and grasps were constructed from independent contact quality spaces. These two changes greatly improved the practicality of the simple algorithm first presented in Section 5.6. This section shows that a significant increase in execution speed can be obtained by parallelizing parts of this algorithm. The algorithm lends itself naturally to parallelization over workspace configurations, which can either represent configurations of links of the hand or configurations of the robot wrist. The computation cannot be completely parallelized, however, as the paragraphs below explain. In particular, the process of chaining together the best links of the hand to form good grasps requires local communication between processors in a parallel machine.

5.8.1 Notation for Parallel Algorithms

Some standard notation is necessary to indicate that a portion of an algorithm can be parallelized, and to represent the process of local communication.

Much of the computation in Algorithm 5.2 is repeated over all configurations of a link or over all configurations of the wrist. Ideally, there would be enough processors in a massively parallel computer to assign each small chunk of configuration space to a unique processor in the computer. If this were the case, the loop could be completely replaced with a single parallel operation. This ideal transformation can be written as follows:

```
for config in workspace_configs() do
   result[config] := process_config(config);
```

—becomes—

```
for i in processors() in parallel do
begin
   config[i] := workspace_config(i);
   result[i] := process_config(config[i]);
end
```

If there are enough processors, then each **config** of the original loop is allocated to a processor **i** in a parallel machine. This results in an array of **config** values that can be indexed by processor number. Where array **result** was computed in the original loop by running **process_config** for each **config** sequentially, **result** is computed in the second loop by running **process_config** on all processors in parallel.

If there are not enough processors to cover the entire set of samples returned by function **workspace_configs()**, then each processor will represent some number of virtual processors V. Time to execute the loop above will depend on the size of V. If the number of workspace configuration samples is written as R^3O^3, representing three sampled position parameters R^3 and three sampled orientation parameters O^3, and if the number of processors is P, then:

$$V = \left\lceil \frac{R^3O^3}{P} \right\rceil, \tag{5.5}$$

where $\lceil X \rceil$ represents the smallest integer greater than X.

If there are as many processors as configuration space samples, then $V = 1$, and the loop can be executed in a single step. If there is only one processor, then $V = R^3O^3$, and the body of the loop must be repeated for each configuration space sample, as in the serial loop described above.

It is easy to see that when **process_config** is executed on all processors in parallel, the **results** array that is computed will be distributed over all the processors. This is fine when each processor must reference only its own results, but when a processor i needs to reference an intermediate result computed by another processor j, some communication between processors must take place. This communication is represented with a **fetch** function. The **fetch** function is simply a reference to memory that is not local to a processor, and it is used as follows:

```
for config in workspace_configs() do
   transformed_result[config] := compute_transformed_result(config);

for config in workspace_configs() do
begin
   next_config := transform(config);
   result[config] := transformed_result[next_config];
end
```

—becomes—

```
for i in processors() in parallel do
begin
   config[i] := workspace_config(i);
   transformed_result[i] := compute_transformed_result(config[i]);
end

for i in processors() in parallel do
begin
```

```
    next_config[i] := transform(config[i]);
    next_proc[i] := proc_of(next_config[i]);
    result[i] := fetch(transformed_result[next_proc[i]]);
end
```

The example illustrates the following operations:

1. In both versions, a **transformed_result** is computed over the entire set of **workspace_configs()**.
2. In both versions, final array **result** is defined to be a transformed version of **transformed_result**. The transformation function maps one configuration to another and is given by **transform(config)**.
3. The serial version simply looks up the appropriate configurations of **transformed_result** and stores them in **result**.
4. In the parallel version, the required information from the **transformed_result** is in general not available in internal processor memory. The processor where the information can be found can be determined, however, using function **proc_of(next_config[i])** to obtain processor **next_proc[i]**. Each processor *i* must communicate with processor **next_proc[i]** to get the value of the **transformed_result** array that should be stored in processor *i* as the value of **result[i]**. The required communication operation is performed by the **fetch** function.

In general, performing **fetch** operations will be slow. Parallel computation is much more efficient if all processing can be done locally than if the processors must communicate. When there is communication, it is most efficient if this communication takes place over short distances. These efficiency concerns are reviewed in detail when a machine architecture is outlined in Chapter 6.

5.8.2 Algorithm Overview

The algorithm described in this section is a direct parallelization of function **lower_bound_gq_1**. The pseudocode algorithms for this parallel version can be found in Appendix A, beginning with top level function **lower_bound_gq_1_parallel**. The interesting parts of this revised algorithm are displayed in the text that follows. The parts of this algorithm that lend themselves to parallelization are computation of contact quality measures over the space of link configurations, computation of optimal finger subchains over the space of link configurations, and the merging of optimal fingers to form optimal grasps over the space of wrist configurations. Parallelization of these computations is illustrated, and time estimates are provided below.

5.8.3 Computing Contact Quality Space

Contact quality space can be computed over all link configurations in parallel. This computation can be done independently; a processor representing a link configuration does not

need to reference information stored at processors representing other link configurations. This means that the loop:

```
for config in workspace_configs() do
   cq[link, config] :=
      compute_link_cq_1
         (link, config, orient, contact_assignment, prototype, hand_model,
            object, obstacles);
```

of function compute_cq_1 can be replaced by:

```
for i in processors() in parallel do
begin
   config[i] := workspace_config(i);
   cq[link, i] :=
      compute_link_cq_1
         (link, config[i], orient, contact_assignment, prototype, hand_model,
            object, obstacles);
end
```

in parallel function compute_cq_1_parallel.

Time

Parallel computation of the contact quality array over the space of link configurations results in a linear speedup proportional to the number of processors available. Because this part of the problem can be broken into small, independent chunks, it is well-suited for this type of parallel computation. In Section 5.7.2, the following time estimate was stated for the contact quality calculation:

$$O(L(f_E f_L + f_T f_L)), \tag{5.6}$$

sampled over

$$\Re^3 \times SO(3). \tag{5.7}$$

Because calculation of each contact quality measure can be done independently, any ratio of configuration space samples to processors can be effectively exploited. To capture the variation in time to compute the contact quality array with the number of processors available, both the number of configuration space samples and the number of processors must be specified. If the number of configuration space samples is described as R^3O^3 and the number of processors is P, the time estimate for this algorithm can be rewritten as follows:

$$O\left(L\left\lceil\frac{R^3O^3}{P}\right\rceil(f_E f_L + f_T f_L)\right). \tag{5.8}$$

5.8.4 Estimating Grasp Quality Space

The space of grasp quality measures can be computed over all wrist configurations in parallel, but some amount of communication is required. Once optimal finger quality measures have been computed, the combination of these optimal finger quality measures to form optimal grasps can be parallelized over the space of wrist configurations. In particular, the loop of function **compute_gq_1**:

```
for wrist_config in workspace_configs() do
begin
   {minimize grasp quality gq over finger qualities}
   gq[wrist_config] := MAX_QUALITY;
   for finger in fingers_of(hand_model) do
   begin
      prox_config := prox_link_config_of(finger, hand_model, wrist_config);
      if fq[finger, prox_config] < gq[wrist_config] then
         gq[wrist_config] := fq[finger, prox_config];
   end
end
```

can be replaced by the parallel loop:

```
for i in processors() in parallel do
begin
   wrist_config[i] := workspace_config(i);
   {minimize grasp quality gq over finger qualities}
   gq[i] := MAX_QUALITY;
   for finger in fingers_of(hand_model) do
   begin
      prox_config[i] := prox_link_config_of(finger, hand_model, wrist_config[i]);
      prox_proc[i] := proc_of(prox_config[i]);
A     q[i] := fetch(fq[finger, prox_proc[i]]);
      if q[i] < gq[i] then
         gq[i] := q[i];
   end
end
```

in function **compute_gq_1_parallel**.

Notice the **fetch** function required in the parallel loop (A). This indicates the fact that there exists a transformation function mapping the wrist configuration to the configuration of the proximal link of each finger. Optimal finger information for the fingers of the hand

is distributed over different processors representing the proximal link configurations of the fingers, and it must be transferred to the common processor representing a wrist configuration before it can be merged to generate an optimal grasp quality measure for the entire grasp. This transfer step is achieved using communication from processor to processor in the parallel machine.

Quality of a Finger

The parallel function to construct the space of optimal finger quality measures can be parallelized over the space of link configurations, but it requires a significant amount of processor to processor communication. Here, the loop in function **finger_quality_1**:

```
for config in workspace_configs() do
begin
   {maximize next link subchain quality over joint angles}
   max_sq := MIN_QUALITY;
   for theta in joint_angles_of(link, hand_model) do
   begin
      next_config := next_link_config_of(theta, link, hand_model, config);
      if subchain_quality[next_config] > max_sq then
         max_sq := subchain_quality[next_config];
   end

   {take minimum of link and current subchain quality values}
   fq[config] := min(cq[link, config], max_sq);
end
```

can be replaced by:

```
for i in processors() in parallel do
begin
   config[i] := workspace_config(i);
   {maximize subchain_quality over joint angles}
   max_sq[i] := MIN_QUALITY;
   for theta in joint_angles_of(link, hand_model) do
   begin
      next_config[i] := next_link_config_of(theta, link, hand_model, config[i]);
      next_proc[i] := proc_of(next_config[i]);
A     sq[i] := fetch(subchain_quality[next_proc[i]]);
      if sq[i] > max_sq[i] then
         max_sq[i] := sq[i];
```

```
    end

    {take minimum of link and current subchain quality values}
    fq[i] := min(cq[link, i], max_sq[i]);
  end
```

in function **finger_quality_1_p**.

Again, note the **fetch** function in the parallel loop (A). Recall that function **finger_quality_1** operates by iteratively extending the length of all optimal subchains by a single link. Given a **link** and a joint angle **theta**, the processor representing the configuration of the **next_link** can be referenced to check the quality value of the optimal subchain originating at that link. This value will be optimized over all joint angles **theta**. In general, **next_config**, the configuration of **next_link**, will be different than **config**, the configuration of **link**. This means that the **subchain_quality** value found at the processor representing **next_config** must be transferred to the processor representing **config** so that these values can be optimized, and the result stored at a common processor.

Time

The previous sections have described the parallel version of an algorithm that uses an array of contact quality measures for each link of the hand to compute an array of grasp quality measures over the space of wrist configurations. Section 5.7.3 noted that the serial version of this algorithm was dominated by the recursive calls to function **finger_quality_1**. The dominant operation in this function was scanning a range of joint angles for each configuration of each link, resulting in a time estimate of:

$$O(L), \tag{5.9}$$

sampled over

$$\Re^3 \times SO(3) \times S^1. \tag{5.10}$$

The parallel version of this algorithm is very similar to the serial version. Time is dominated by the recursive calls to **finger_quality_1_p**. This function is eventually called for all links of the hand, and a function call involves a scan over a range of joint angles. Configurations of each link are processed in parallel, resulting in a potential speedup of a factor of P, the number of processors, as with the computation of contact quality in Section 5.8.3. This speedup will not be completely realized, however, because of the communication required to find the quality of the optimal subchain corresponding to a link configuration. This communication step is represented by the **fetch** function, and the time required to execute this function depends on the exact mapping of configuration space onto a processor grid. Because of this dependency, the time required to compute the grasp quality space is not formally analyzed until Chapter 6.

It is possible to provide a time estimate that leaves the complexity of the **fetch** command as an expression yet to be determined. As in Section 5.8.3, the number of configuration space samples is defined as R^3O^3. In addition, the size of a joint angle range is represented by d_{angle}. Then time can be estimated as:

$$O(L\left\lceil\frac{R^3O^3}{P}\right\rceil(Od_{angle})F). \tag{5.11}$$

Time is proportional to the number of links of the hand L, the number of virtual processors each physical processor must simulate $\left\lceil\frac{R^3O^3}{P}\right\rceil$, and the number of joint angle samples Od_{angle}, which depends both on the number of samples taken of each orientation parameter (roughly O) and on the size of a joint angle range. Total time is also proportional to time to execute the **fetch** function, represented here as F. Parameter F is dependent on the number of configuration space samples and the number of processors, as will be seen in Chapter 6.

5.8.5 Summary

This section has shown that the execution time of the algorithm of Section 5.7 can be reduced by parallelizing this algorithm over the six-dimensional space of link or wrist configurations. The decrease in execution time that is obtained is not a simple linear function of the number of processors used, however. This is due to the processor-to-processor communication required when chaining together good link configurations to form grasp quality estimates for complete grasps. This issue is analyzed further when a parallel machine architecture is specified in Chapter 6.

5.9 Summary and Discussion

This chapter addressed the problem of adding the constraints of hand kinematics and geometry and environment geometry to the search for a good grasp. It derived a parallel algorithm for computing a representation of the space of good grasps. This algorithm made use of a grasp prototype to localize contact quality computations to individual contacts of a grasp. Contact quality measures could be precomputed over the space of link configurations, and the best quality link configurations could be chained together in kinematically feasible ways to produce a space of grasp quality estimates.

Grasp quality estimates are computed by this algorithm over the space of wrist configurations, not over the complete space of hand configurations. While this results in a solution space that can actually be computed, it also means that some mechanism must be provided to extract optimal hand configurations from this projective solution space. It is very easy to provide access to a single optimal hand configuration at each point in this space. The joint angles and link-target object contacts corresponding to an optimal solution can be propagated from the fingertips to the wrist along with the optimal grasp quality estimates. If intermediate subchain quality measures and intermediate optimal

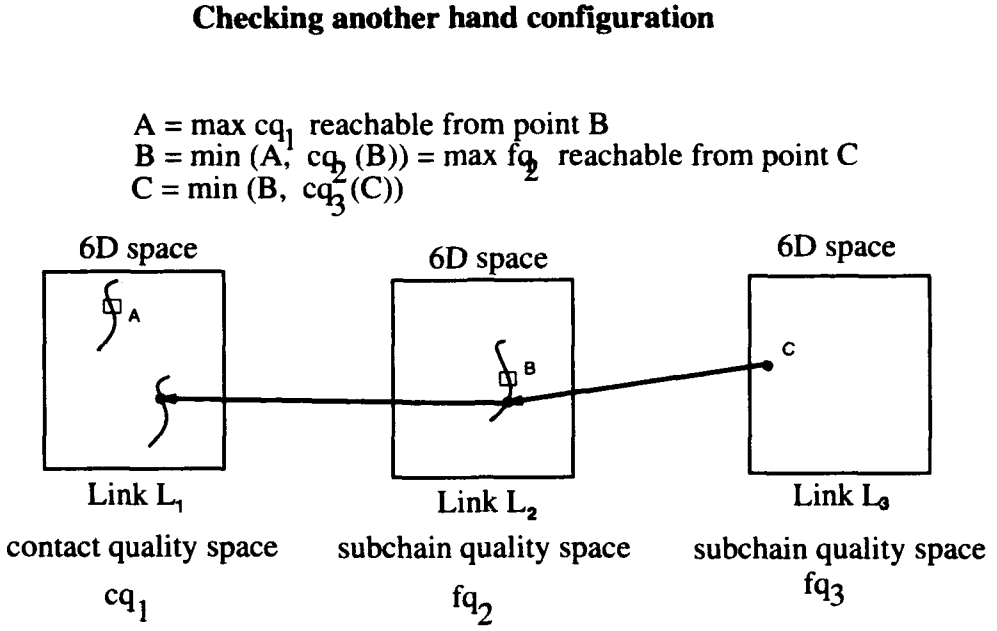


Figure 5.12: The complete space of configurations of a finger can be explored by chaining back through the contact quality spaces of the links of that finger. This figure shows exploration of one finger configuration that is not the optimal configuration passing through points A, B, and C.

solution information are retained, however, it is possible to quickly and easily reconstruct the grasp quality estimate associated with any hand configuration. This means that it is easy to search through the entire space of hand configurations using the optimal grasp quality estimates to highlight regions of good solutions. Figure 5.12 shows the process of working backward through the layers of partial solutions to obtain a complete hand configuration. This process requires only a sequence of memory references. No additional quality or collision information need be computed.

The parallel algorithm presented in this chapter is much more tractable than the simple brute force approach described initially, but it is still too difficult to execute on today's computers. The next chapter shows some ways in which this algorithm can be optimized to fit a particular parallel machine architecture, and some tradeoffs that can be made to tune the algorithm still further to the number of processors and amount of memory on a given parallel machine.

Chapter 6

Optimizing Grasps: Making it Work

The previous chapter described a parallel algorithm for computing a space of grasp quality estimates. This algorithm used a grasp prototype as defined in Chapter 3 to compute contact quality measures for all configurations of all links of the hand, and then used these contact quality measures to chain the best links together to form complete grasps. Kinematics and geometry of the hand, and geometry of the environment were considered when constructing the grasp quality space. The optimal grasp quality estimates were computed for all configurations of the robot wrist, resulting in a solution space that is a projection of the more complete space of hand configurations.

Both the ability to compute and reuse partial solutions represented by the array of contact quality measures and the decision to compute a projective solution space have made it feasible to search through a large space of possible grasps. A number of incremental improvements are required to make this algorithm practical, however, and it will be necessary to tune the algorithm to fit the hardware available. This chapter describes the following issues:

1. **Too few processors:** The number of processors available does not allow each processor to be assigned to a small chunk of configuration space. How should the spaces of link and wrist configurations be distributed over the processor network?

2. **Too little memory:** The amount of on-processor memory limits the accuracy that can be obtained using the current, breadth-first algorithm. Should this memory-intensive breadth-first approach be used, or will an alternative, time-intensive depth-first approach be more effective?

3. **Too little time:** The amount of time available for executing this algorithm limits the density at which hand configuration space can be sampled. If the number of samples is small, but these samples are accurate, good solutions may be missed. How should the desire to test accurate hand configurations be balanced with the desire to sample the entire hand configuration space?

The first issue listed above involves choosing a particular mapping of configuration space onto a network of processors. There are not enough processors available to represent an entire six-dimensional configuration space grid, and so Section 6.1 begins by describing the three-dimensional processor network assumed in this chapter, and Section 6.2 describes a mapping of configuration space onto this processor network. Section 6.3 uses the given processor network and mapping to reevaluate the time complexity of the parallel algorithm used to compute an array of grasp quality estimates in Chapter 5. In particular, this section describes one implementation of the function used to **fetch** information from one processor into another. This elaboration of the **fetch** function leads to two improvements in the parallel algorithm: Section 6.4 describes how the computation of finger quality arrays can be made more efficient, and Section 6.5 describes how a simple distributed representation of environment and target object geometries can be computed and used to efficiently construct link contact quality measures.

The second and third issues listed above involve tradeoffs that can be made to balance memory, time, and accuracy requirements. Section 6.6 describes an alternate algorithm that reduces memory requirements at the cost of execution time. Section 6.7 shows how coverage of configuration space can be maintained while this space is sampled coarsely. This allows time to be gained by sacrificing accuracy. The actual balancing of the tradeoffs described in Sections 6.6 and 6.7 is performed in Chapter 7, which describes the final implementation of this parallel algorithm and presents examples.

6.1 A Specific Processor Network

The previous chapter presented a parallel algorithm for computing a space of grasp quality measures. This algorithm was written to sample a six-dimensional space of link or wrist configurations in parallel. To completely distribute a sampled six-dimensional space over an array of processors would require an enormous number of processors. For example, taking 32 samples of each of the 6 parameters would result in approximately one billion grid points. A machine with one billion processors is not practical, and this section describes the much smaller processor network assumed in this chapter.

The processor network assumed in this chapter can be described very simply. The following assumptions are made:

- there is a three-dimensional grid of processors, and
- each processor is connected directly to its neighbors in all three dimensions (there are six neighbor connections per processor).

A three-dimensional grid of processors represents a much more feasible assumption than a six-dimensional grid of processors. If each of the 3 parameters is sampled 32 ways, approximately 32,000 processors will be required. Machines of this size already exist. The Thinking Machines CM200, for example, comes in a 64,000 processor version.

6.2 Mapping Configuration Space onto the Processor Grid

The processor topology assumed in this chapter is a three-dimensional grid of processors, where each processor is connected directly to its neighbors for fast local communication in three dimensions. This section describes how configuration space is mapped onto this gr'd, and it shows how the parallel algorithm of the previous chapter is revised to make this mapping explicit.

6.2.1 Representing Position Space in Parallel

This report assumes that position space is distributed over a three-dimensional grid of processors. Each processor within this grid represents a three-dimensional chunk of Cartesian space. This proves to be a useful representation because collision and contact information is mostly position-based. The environment and target object geometry can be preprocessed and distributed over the processor network to simplify identification of collision and contact conditions. For example, simple space filling algorithms can be used to mark the processors as inside an object or within free space. Once a processor has been marked as inside an object, the space it occupies cannot be occupied by any part of the hand in any collision-free grasp. This local geometric information can be used many times as the portion of space represented by a processor forms a part of many different hand configurations. Section 6.5 below describes how this and similar information is computed and used.

One problem with the decision to represent the three-dimensional position space in parallel is that the quality arrays for links, subchains, fingers, and complete grasps continue to be computed over the full six-dimensional configuration space. It is necessary to determine where these six-dimensional results will be stored. The obvious solution is to assume that each processor will store a full three-dimensional space of orientations in internal processor memory. Unfortunately, this solution is not currently practical due to the large memory requirements associated with the approach, and the parallel algorithms will be revised so that this large amount of memory is not required (Sections 6.6 and 7.2). For the next few sections, however, it is assumed that each processor in the three-dimensional position space grid is capable of storing a full three-dimensional orientation map.

6.2.2 A Revised Parallel Algorithm

The paragraphs above noted that in this chapter the grasp quality computation is parallelized over position space only. It is useful to rewrite the parallel algorithm of the previous chapter to explicitly reflect this partitioning of configuration space loops into serial and parallel components. The revised algorithm can be found in Appendix A, beginning with top-level function **lower_bound_gq_2_parallel**. The main difference from the parallel algorithms of the previous chapter is that the loop:

for i in processors() in parallel do

has been replaced by a dual loop, such as:

```
for orient in workspace_orients() do
   for i in processors() in parallel do
```

In addition, arrays that were previously indexed only by a processor in the original parallel algorithm beginning with function **lower_bound_gq_1_parallel**, when a processor represented a chunk of the full configuration space, are now indexed by both an orientation parameter and a processor, because each processor now represents a chunk of position space. For example, in function **compute_cq_2_parallel**, contact quality array **cq**, previously indexed by a **link** and a processor **i**, is now indexed by a **link**, a **link_orient**, and a processor **i**:

```
cq[link, link_orient, i] :=
   compute_link_cq_1
      (link, link_config[i], orient, contact_assignment, prototype, hand_model,
         object, obstacles);
```

6.3 Estimating Time Complexity of the Algorithm

The execution time estimates presented for the parallel algorithm of the previous chapter can be refined to reflect the processor topology and the mapping of configuration space onto the processor grid described above. This section outlines these refinements. In particular, it describes one implementation of the **fetch** command used for local communication in these parallel algorithms (see functions **compute_gq_2_parallel** and **finger_quality_2_p**). This description of the implementation of the **fetch** command will lead to two improvements to the algorithm, which are described in Sections 6.4 and 6.5.

6.3.1 Computing Contact Quality Space

The body of top-level function **lower_bound_gq_2_parallel** consists of computing a contact quality array and then using this contact quality array to compute an array of optimal grasp quality estimates over the space of wrist configurations. Chapter 5 showed that the time to compute the contact quality array, that is, the time to execute function **compute_cq_1_parallel** (found in Appendix A) could be estimated as:

$$O\left(L\left\lceil\frac{R^3O^3}{P}\right\rceil(f_E f_L + f_T f_L)\right). \tag{6.1}$$

Because only position space is distributed over the processor grid in this chapter, a more accurate description of the time required to execute function **compute_cq_2_parallel** (Appendix A) is:

$$O\left(LO^3\left\lceil\frac{R^3}{P}\right\rceil(f_E f_L + f_T f_L)\right). \tag{6.2}$$

This captures the fact that contact quality is computed over all links of the hand L and over all orientations of those links O^3. It captures the fact that each physical processor must duplicate all calculations over $\left\lceil\frac{R^3}{P}\right\rceil$ virtual processors. It also includes the complexity of computing a single contact quality measure, which depends on the number of features in the environment f_E and the number of features of the target object f_T, as well as the number of features in a link of the hand f_L.

6.3.2 Estimating Grasp Quality Space

The analysis of the time required to compute the contact quality array is very simple, because the contact quality computation is a local computation that can be completely parallelized over configuration space. Execution time for the contact quality computation depends linearly on the ratio of configuration space samples to processors.

Analysis of the time required to combine contact quality measures to produce a grasp quality array is not as simple. This is due to the local communication required to transfer subchain quality information from one link to the next. The grasp quality computation is represented by function **compute_gq_2_parallel** in Appendix A. Time to execute this function is dominated by the recursive calls to function **finger_quality_2_p**. This function is eventually called for all links of the hand L. Time to execute each function call is dominated by the time to repeat the **fetch** function over the sampled range of joint angles, represented by Od_{angle}, for all virtual processors represented by a single physical processor $\left\lceil\frac{R^3O^3}{P}\right\rceil$. In Chapter 5, the **fetch** function could not be fully described, because the processor topology was not known, and the end of that chapter stated the following expression for time complexity of the grasp quality computation:

$$O(L\left\lceil\frac{R^3O^3}{P}\right\rceil(Od_{angle})F). \tag{6.3}$$

Now that the algorithm has been rewritten to operate serially on link orientations, Expression 6.3 can be rewritten as:

$$O(LO^3\left\lceil\frac{R^3}{P}\right\rceil(Od_{angle})F). \tag{6.4}$$

Expression F, representing complexity of the **fetch** function, was left undetermined in Chapter 5. This section describes one implementation of the fetch function, allowing derivation of a value for F.

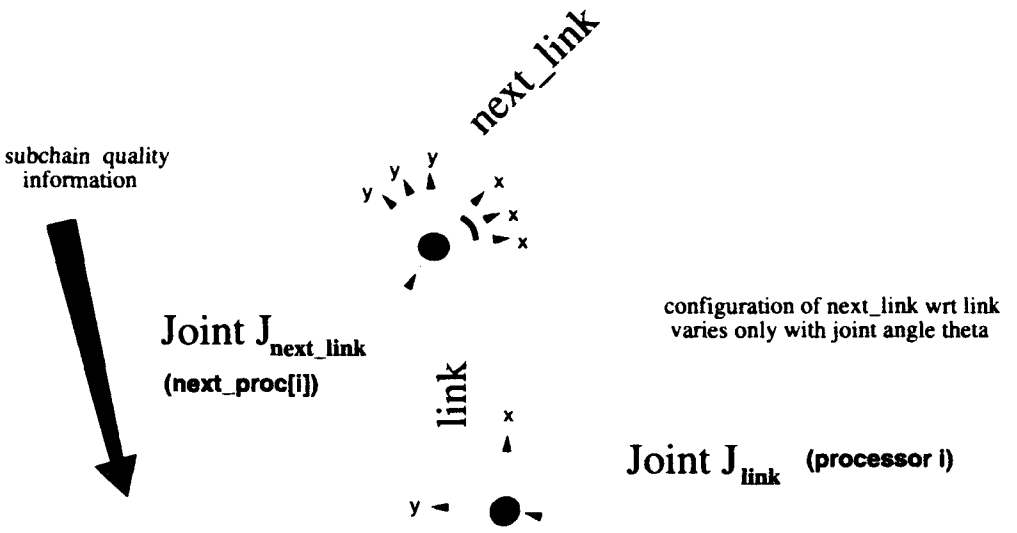


Figure 6.1: The relationship between the coordinate systems of **link** and **next_link** is a function of joint angle. Subchain quality information is passed from the processor representing the configuration of **next_link** to the processor representing the configuration of **link**.

Fetching Information into a Single Processor

The **fetch** function was originally described as a very general function that played the role of transferring information from one processor to another. In the parallel algorithm described in Section 5.8, however, the **fetch** function plays a much more specific role. This section describes that role for the task of fetching information into a single processor, and shows how this command might be implemented. The following section describes how this implementation can be parallelized, so that the **fetch** command can be used to bring information into all processors at once.

The **fetch** function is referenced in functions **compute_gq_2_parallel** and **finger_quality_2_p** (Appendix A). The analysis of the operation of this function is similar for either reference, and so only one reference will be examined. The following excerpt shows how the **fetch** function is used in function **finger_quality_2_p**:

```
{get subchain_quality for current link}
for link_orient in workspace_orients() do
   for i in processors() in parallel do
   begin
      config[i] := workspace_config_2(link_orient, i);
      {maximize subchain_quality over joint angles}
```

```
        max_sq[i] := MIN_QUALITY;
        for theta in joint_angles_of(link, hand_model) do
        begin
            next_config[i] := next_link_config_of(theta, link, hand_model, config[i]);
            next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
            next_proc[i] := proc_of(next_config[i]);
            sq[i] := fetch(subchain_quality[next_orient, next_proc[i]]);
            if sq[i] > max_sq[i] then
                max_sq[i] := sq[i];
        end

        {take minimum of link and current subchain quality values}
        fq[link_orient, i] := min(cq[link, link_orient, i], max_sq[i]);
    end
```

This excerpt shows the construction of finger quality array **fq**, which updates the subchain quality measures of array **subchain_quality** so that they represent subchains one extra link in length. To perform this operation, the function must reference the link contact quality array **cq** for the current **link** and **config** and the quality measures of the **subchain_quality** array for reachable configurations of the distal neighbor of **link**, given parameter **config**. Because position space is distributed over the processor grid, a processor will find that it must reference elements of the **subchain_quality** array that are not present in local memory. Making these non-local memory references is the role of the **fetch** function.

This section describes a serial call to the **fetch** function. In other words, it describes what happens when a single processor makes one of these non-local memory references. When the **fetch** command is called, the following parameters have been specified:

link	=	a link of the hand
i	=	processor representing the position of **link**
link_orient	=	orientation of **link**
theta	=	joint angle of the distal joint of **link**

From this information, the following additional parameters can be computed:

next_link	=	the distal neighbor of **link**
next_proc[i]	=	processor representing the position of **next_link**, calculated from **i** and **link_orient**
next_orient	=	orientation of **next_link**, calculated from **link_orient** and **theta**

The goal of the **fetch** function is to move **subchain_quality[next_orient]** from processor **next_proc[i]** to processor **i**. The processor representing a link is determined by the position of that link. The local coordinate system of each link is defined to have its origin at the proximal joint of that link as shown in Figure 6.1. Given a position and orientation for

link, the position and orientation of **next_link** varies only with joint angle **theta** (the joint angle of joint J_{next_link} in Figure 6.1). The goal is to collect the maximum value of the **subchain_quality** array for configurations of **next_link** over the entire joint angle range. A call to the **fetch** function is used to collect each non-local **subchain_quality** value.

In this chapter, each processor represents a particular link position, and it stores an array of **subchain_quality** values over a space of link orientations. This means that the desired **subchain_quality** value must be extracted from the **subchain_quality** array on the processor representing the position of **next_link** and transferred to the processor representing the position of **link**. To move the **subchain_quality** value between these two processors, a *routing-path* is constructed along the axis of **link**, as shown in Figures 6.2 and 6.3. The desired **subchain_quality** value that begins in processor **next_proc[i]** is passed from neighbor to neighbor according to the relative motion commands in the routing path, ending up in processor **i**.

The serial operation of the **fetch** command is summarized as follows:

1. Parameter **next_orient** is calculated from joint angle **theta**.
2. Parameter **subchain_quality[next_orient]** is extracted from processor **next_proc[i]**.
3. This value is routed to processor **i**, following the routing path for the current **link** and **link_orient**.
4. This value is stored in temporary variable **sq** on processor **i**.

The **fetch** command is repeated for all **theta** values within the joint angle range of joint J_{next_link}, and the maximum **sq[i]** value that results is kept in temporary variable **max_sq[i]**. The new subchain quality entry **fq[link_orient, i]** is set to be the minimum of **max_sq[i]** and local link contact quality value **cq[link_orient, i]**.

Fetching Information into All Processors

The previous section described how the **fetch** command works for moving subchain quality information from one specified link configuration to another. But the **fetch** command is meant to be executed for all processors, or all link positions, in parallel. This section describes how this parallel operation works.

It is easy to parallelize this **fetch** command over the space of link positions, because the direction in which information must travel at each step is the same for all processors. Figure 6.4 shows the information transfer required for three processors in the example two-dimensional grid. This figure shows that the routing paths all have the same shape. This is true because each routing path is derived from the shape of the current link and the orientation of that link. It does not depend on link position, and so it is the same for all processors **i**. In other words, the relative position of **next_proc[i]** with respect to **i** is the same for any processor **i**. This means that **subchain_quality[next_orient]** can be extracted from all processors **next_proc[i]** in parallel. It can be routed in parallel from processor to processor using the routing path of Figure 6.3, and the subchain quality information will end up in the appropriate processors **i**.

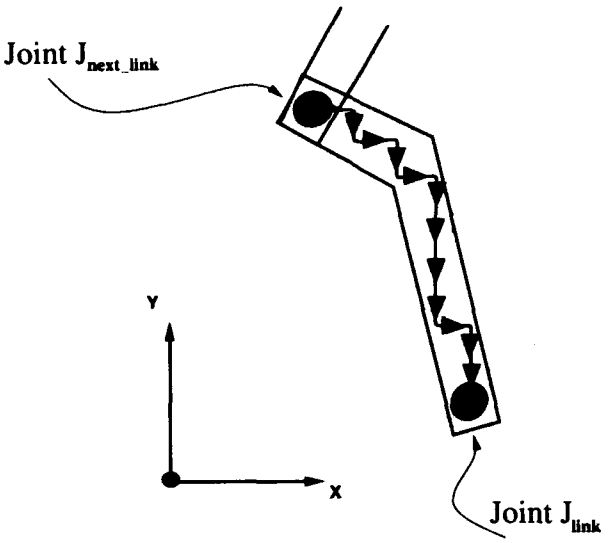


Figure 6.2: Routing information from the distal joint of a link to the proximal joint of that link along the link axis.

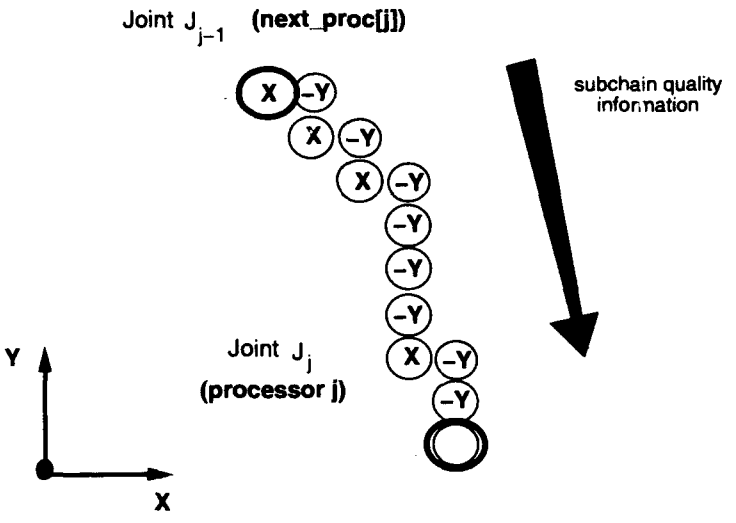


Figure 6.3: A map to direct the routing of subchain quality information along a link axis.

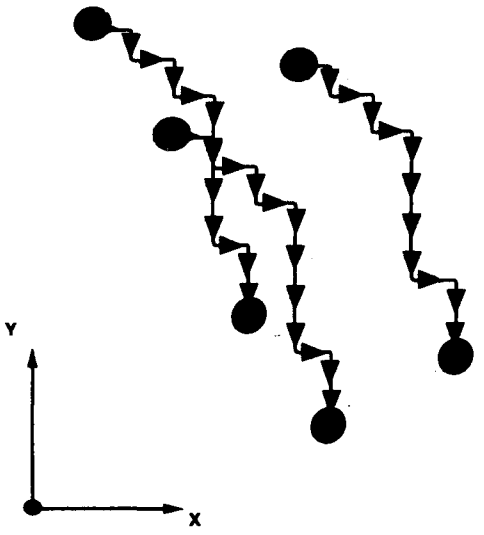


Figure 6.4: Routing paths for three processors in a grid. The vector representing travel from the distal joint of the link to the proximal joint of that link is the same for any processor representing the proximal joint of the link.

It is clear from the routing path that there is a one-to-one mapping from processors i to processors **next_proc[i]**. No two processors i and j will try to fetch information from the same processor **k** during the same instruction. In other words,

$$(\mathsf{i} \neq \mathsf{j}) \Longrightarrow (\mathsf{next\_proc[i]} \neq \mathsf{next\_proc[j]}).$$

Furthermore, as long as all routing is synchronized, that is, as long as all processors complete one step of the routing path before any processors initiate the next step of the routing path, there will be no routing collisions. Each processor will send and receive exactly one message as each step along the routing path is executed.

It is clear that this process takes time proportional to the length of the routing path. The length of the routing path grows with the length of a link d_{link} and with the number of processors intersected by a line of unit length. This can be approximated as $\left\lceil \frac{P^{\frac{1}{3}}}{S} \right\rceil$, where S represents the size of a side of one dimension of the position space cube, and this dimension is sampled by $P^{\frac{1}{3}}$ processors. This gives the following expression for time complexity of the **fetch** function previously described as F:

$$F = \left\lceil \frac{d_{link} P^{\frac{1}{3}}}{S} \right\rceil. \tag{6.5}$$

Combining Expressions 6.4 and 6.5 results in an estimate for time to execute function **compute_gq_2_parallel**:

$$O\left(LO^3 \left\lceil \frac{R^3}{P} \right\rceil (O d_{angle}) \left\lceil \frac{d_{link} P^{\frac{1}{3}}}{S} \right\rceil \right). \tag{6.6}$$

If there are more processors P, then more of the position space can be processed in parallel, which makes term $\left\lceil \frac{R^3}{P} \right\rceil$ smaller, but term $\left\lceil \frac{d_{link} P^{\frac{1}{3}}}{S} \right\rceil$ is somewhat larger, as messages will have to travel a greater distance through the processor grid to get from one link to the next.

6.4 Improving the Finger Quality Computation

The previous section described how the **fetch** command used in function **finger_quality_2_p** is implemented. Given this implementation, the process of maximizing subchain quality over a range of joint angles can be made much more efficient.

Figure 6.1 shows that the following operation is performed repeatedly for joint angles **theta** sampled over the joint angle range of joint J_{next_link}:

- Calculate **next_orient** from **link_orient** and **theta**.
- Look up variable **subchain_quality[next_orient]** in processor **next_proc[i]**.

- Transfer this variable to processor i along the axis of **link**.
- If this variable is greater than **max_sq** in processor i, store this variable in **max_sq**.

The major inefficiency found in this loop is that an expensive processor to processor communication step is performed only to maximize the transferred values at the destination. This section simply rewrites the procedure described above to maximize the **subchain_quality** values at the source, or the distal joint of **link**, and then transfer only the maximum value along the link axis to the proximal joint of **link**. This transformation is possible because the algorithm is parallelized over positions only and because the coordinate system of a link is located at the proximal joint of that link. Although the configuration of **next_link** changes with joint angle **theta**, its position does not change. This means that all **subchain_quality** values stored for **next_link** configurations reachable from a given **link** configuration over a range of joint angles can be found within a single processor, **next_proc[i]**.

This communication inefficiency has been removed in function **finger_quality_2_p_better** (Appendix A), where the following code:

```
for theta in joint_angles_of(link, hand_model) do
begin
   next_config[i] := next_link_config_of(theta, link, hand_model, config[i]);
   next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
   next_proc[i] := proc_of(next_config[i]);
   sq[i] := fetch(subchain_quality[next_orient, next_proc[i]]);
   if sq[i] > max_sq[i] then
      max_sq[i] := sq[i];
end

{take minimum of link and current subchain quality values}
fq[link_orient, i] := min(cq[link, link_orient, i], max_sq[i]);
```

has been replaced by:

```
for theta in joint_angles_of(link, hand_model) do
begin
   next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
   if subchain_quality[next_orient, i] > max_sq[i] then
      max_sq[i] := subchain_quality[next_orient, i];
end

{fetch maximum subchain quality value}
next_pos[i] := next_link_pos_of(link, hand_model, config[i]);
```

```
next_proc[i] := proc_of_2(next_pos[i]);
sq[i] := fetch(max_sq[next_proc[i]]);

{take minimum of link and current subchain quality values}
fq[link_orient, i] := min(cq[link, link_orient, i], sq[i]);
```

Note that the **fetch** command has been extracted from the loop over joint angles **theta**. The values of **subchain_quality[next_orient]** are first maximized over **theta**, and this maximum value is passed from joint J_{next_link} up to joint J_{link}.

Because communication is expensive, function **finger-quality-2-p-better** will be significantly faster than **finger-quality-2-p**. The time complexity estimate can be rewritten as follows:

$$O\left(LO^3\left\lceil\frac{R^3}{P}\right\rceil\left(Od_{angle}+\left\lceil\frac{P^{\frac{1}{3}}d_{link}}{S}\right\rceil\right)\right). \tag{6.7}$$

Compare this to Expression 6.6. The local memory references over a range of joint angles, represented by Od_{angle}, have been separated from the transfer of subchain quality values through the processor grid, represented by $\left\lceil\frac{P^{\frac{1}{3}}d_{link}}{S}\right\rceil$.

6.5 Improving the Link Quality Computation

The previous section showed a way in which the grasp quality computation could be made more efficient by reducing the amount of communication required between processors. This section presents a technique used to speed the contact quality computation by doing a one-time reduction of environment and target object geometry into a distributed representation of local geometric information.

If a simple approximation of the geometry of each link of the hand can be made, then local geometric information can be computed and stored at each processor. It can then be combined to reconstruct link and subchain contact quality measures as needed. These contact quality measures can be reconstructed by collecting local geometric information when routing subchain quality information along a link axis as shown in Figure 6.2. This avoids repetition of the time-consuming collision and contact calculations that were previously performed for each link configuration (see Algorithm 5.4, reference points A and C). The next few paragraphs show how this process works.

6.5.1 Computing Local Geometric Information at a Point

Local geometric information can be distributed over the grid of processors representing position space. This requires local representations of link geometry on a processor grid. A very simple representation is used here:

- The geometry of a link in a given configuration is approximated with a set of spheres centered at the processors located on the central axis of that link.

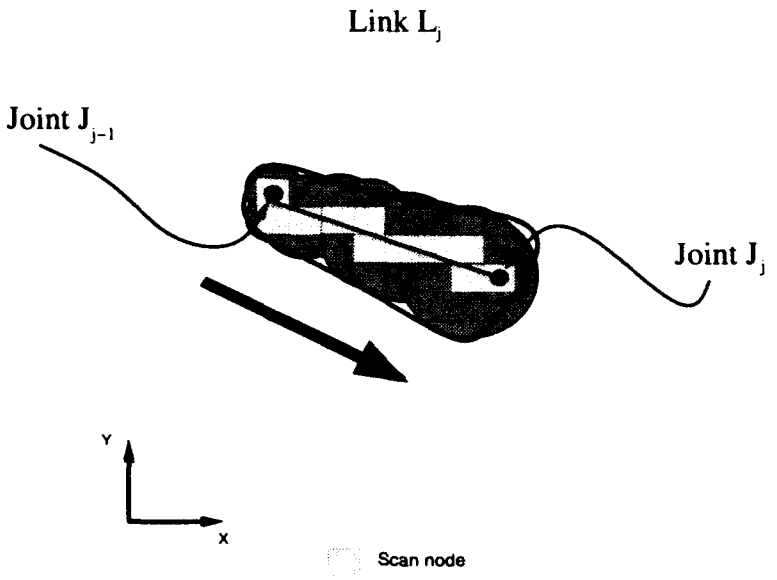


Figure 6.5: Collecting contact quality information along a link axis. The link geometry can be approximated as a sequence of spheres centered at grid points along the link axis.

A link approximated in this way, as a set of spheres centered on the link axis, might have a cross section like that shown in Figure 6.5. In this figure, the link boundary is outlined in bold, and a line has been drawn to indicate the location of the link axis. The routing path for this link and link configuration runs from joint J_{j-1} to joint J_j along the link axis, passing through the lightly shaded boxes. At each of these *scan nodes*, a darkly shaded circle has been drawn, centered at that node. Together, these shaded circles form an approximation of the entire link geometry. Individually, they form a simple representation of the local link geometry at each processor on the routing path.

The representation of a link as a collection of spheres allows local collision and contact information to be computed very simply. The processor representing each portion of a given link in a given configuration needs to test whether its portion of the link lies in free space and whether it is in contact with the target object. If these two conditions are met, the contact quality measure for the contact can be evaluated. To allow this set of operations to be performed, it is sufficient to store the following geometric information at each processor:

1. **collision_dist** = the nearest distance to a collision with some object in the environment.

2. **contact_dist** = the nearest distance to contact with the target object.

3. **contact_point** = the contact point corresponding to the nearest contact with the target object.

4. **contact_feature** = the contact feature corresponding to the nearest contact with the target object.

As shown in Algorithms 6.2 and 6.3 below, parameters **collision_dist** and **contact_dist** allow computation of whether a portion of a link lies in free space and whether it is in contact with the target object. This requires only a local link diameter, **diam**, which can be precomputed for each step of a routing path. Using parameters **contact_point** and **contact_feature**, Algorithm 6.4 shows how a contact wrench is computed for any contact between a portion of a link and the target object. Algorithm 6.1 pulls these functions together to generate a local contact quality measure that is interpreted as follows:

MIN_QUALITY	$\Longrightarrow$	local collision
MAX_QUALITY	$\Longrightarrow$	no local collision, no contact assignment
0	$\Longrightarrow$	no local collision, contact assignment, no local contact
other	$\Longrightarrow$	no local collision, contact assignment, contact

The paragraphs below describe how this information is used to compute link and subchain contact quality measures.

6.5.2 Collecting Collision and Contact Information for a Link

The previous paragraphs have shown that geometric information describing the environment and the target object can be reduced to a set of local distance measurements and local contact information distributed over a position space grid of processors. The motivation behind computing this information was to speed the computation of link contact quality measures in repeated calls to function **compute_link_cq_1**. Given a link and a configuration of that link, a local contact quality measure, **local_cq**, can be computed for each point along the link axis. This section shows how this local collision and contact information can be collected during a **fetch** command to generate a contact quality measure for a given link and link configuration.

The rules for constructing a link contact quality measure from local contact quality measures are very simple:

- All processors representing points on the link axis must indicate that the link lies in free space at that point. Any collision sets the link contact quality parameter **link_cq** to MIN_QUALITY.
- If there are no collisions, the link contact quality parameter **link_cq** is set to the maximum of all local contact quality values **local_cq[i]**.

The following section of code (similar to a portion of function **link_axis_quality_scan** in Algorithm 6.5) shows how the local contact quality values along a routing path can be used to construct a link contact quality measure:

```
for i in processors() in parallel do
```

```
compute_local_cq
   (diam, collision_dist, contact_dist, contact_point, contact_feature, link, link_orient,
      orient, contact_assignment, prototype)
begin
   local_cq := MIN_QUALITY;
   if no_collisions_local?(diam, collision_dist, contact_dist) then
   begin
      if no_contact_assignment?(link, contact_assignment) then
         local_cq := MAX_QUALITY;
      else
         if no_contact_local?(diam, contact_dist) then
            local_cq := 0;
         else
         begin
            contact := contact_of(link, contact_assignment);
            wrench := contact_wrench_local(contact_point, contact_feature, link_orient);
            local_cq := contact_quality(orient, contact, wrench, prototype);
         end
   end
   return(local_cq);
end
```

Algorithm 6.1:

```
no_collisions_local?(diam, collision_dist, contact_dist)
begin
   if (diam < (collision_dist - EPS_COLLISION)) and
      (diam < (contact_dist + EPS_CONTACT)) then
      return(TRUE);
   else
      return(FALSE);
end
```

Algorithm 6.2:

```
no_contact_local?(diam, contact_dist)
begin
   if (diam < (contact_dist + EPS_COLLISION)) and
      (diam > (contact_dist - EPS_CONTACT)) then
      return(FALSE);
   else
      return(TRUE);
end
```

Algorithm 6.3:

```
contact_wrench_local(contact_point, contact_feature, link_orient)
begin
   if is_edge?(contact_feature) then
      force := edge_inside_normal_of(contact_feature, link_orient);
   else
      {feature has a unique inside normal at the contact point}
      force := inside_normal_of(contact_feature, contact_point);

   dist := subtract(GRASP_CENTER, contact_point);
   torque := scale (cross(force, dist), TORQUE_MULTIPLIER);
   return(build_wrench(force, torque));
end
```

Algorithm 6.4:

```
    link_cq[i] := 0;

  for step in routing_path_of(link, link_orient, hand_model) do
  begin
    diam := diam_of(location_of(step), link, hand_model);
    for i in processors() in parallel do
    begin
      local_cq[i] :=
        compute_local_cq
          (diam, collision_dist_of(lc_info[i]), contact_dist_of(lc_info[i]),
            contact_point_of(lc_info[i]), contact_feature_of(lc_info[i]), link,
            link_orient, orient, contact_assignment, prototype);
      if local_cq[i] = MIN_QUALITY then
        link_cq[i] := MIN_QUALITY;
      if link_cq[i] > MIN_QUALITY then
        link_cq[i] := max(local_cq[i], link_cq[i]);
    end
    link_cq := local_fetch(link_cq, direction_of(step));
  end
```

This operation consists of a set of simple numerical comparisons and simple local communication steps. The numerical comparisons check that neither the local contact quality **local_cq[i]** nor the current cumulative link contact quality **link_cq[i]** are equal to **MIN_QUALITY**. If this is false, **link_cq[i]** is set to **MIN_QUALITY**. If it is true, **link_cq[i]** is set to be the maximum of the two values. Each local communication step (**local_fetch**) shifts the updated **link_cq[i]** parameter one step through the processor grid in the direction given by the routing path (**direction_of(step)**).

The result of this operation is the link contact quality measure, interpreted as:

MIN_QUALITY	$\Longrightarrow$	collision
MAX_QUALITY	$\Longrightarrow$	no collision, no contact assignment
0	$\Longrightarrow$	no collision, contact assignment, no contact
other	$\Longrightarrow$	no collision, contact assignment, contact

An alternative way to represent this computation is to draw a circuit illustrating the **link_cq** value read into a processor, the local operation performed at the processor, and the **link_cq** value sent to the next processor along a routing path. Figure 6.6 shows such a simple circuit. Inputs are on the left, outputs on the right, and information available locally at each processor is shown coming into the top of the circuit. The top figure shows the **FREE?** circuit, which outputs **TRUE** iff the portion of the link seen so far is free of collisions. The bottom figure uses the output of the **FREE?** circuit to switch between an output quality value of **MIN_QUALITY**, which indicates a collision, and the maximum **local_cq** value seen so far.

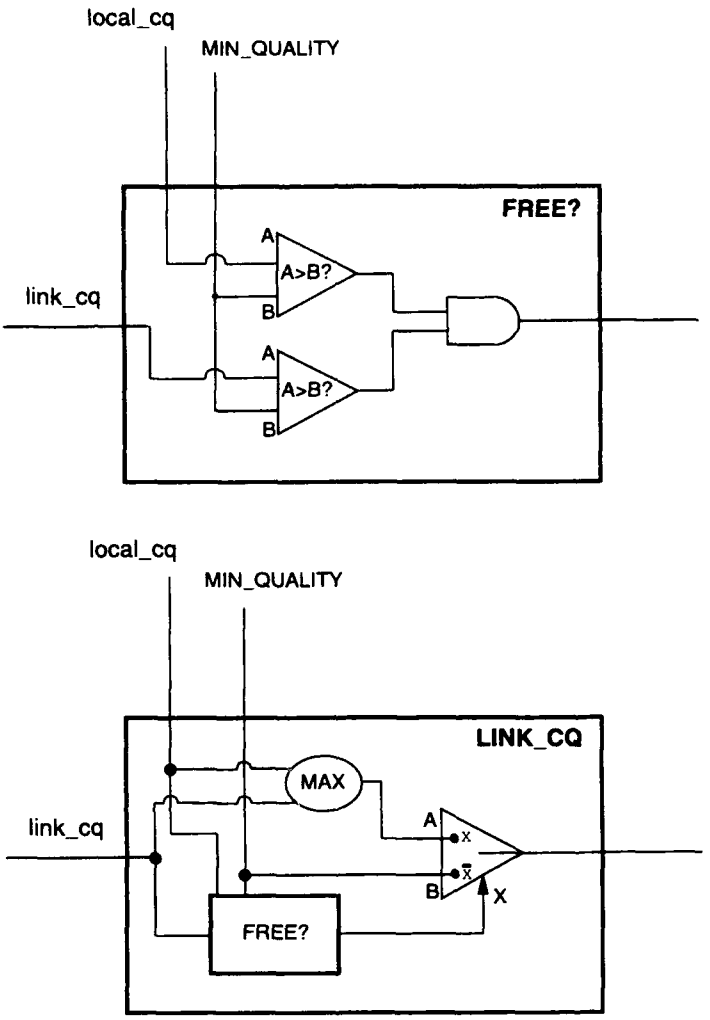


Figure 6.6: The top circuit uses the local contact quality measure **local_cq** and the link contact quality measure **link_cq** to determine whether the portion of the link seen so far lies in free space. The bottom circuit updates the link contact quality measure **link_cq** by taking the maximum of the input **link_cq** value and the local contact quality measure **local_cq** if the link seen so far lies in free space.

The inputs and outputs of the individual processor circuits are wired together as indicated by the routing path for a link and link configuration (Figure 6.7) to obtain a complete circuit that returns **link_cq** for an entire link. The input of the first box is set to zero. The output of the complete circuit is **MIN_QUALITY** if there is a local collision somewhere along the routing path, or it is the maximum of all **local_cq** values along the routing path. If there are no collisions, and no values of **local_cq** are greater than zero, the output will be zero. A strictly positive **link_cq** value is required, however, for the link to be part of a stable grasp.

This **link_cq** parameter can be computed during the **fetch** command used to access optimal subchain values, because both operations require passing information from one joint to the next. Algorithm 6.5 shows the complete scan function. The new set of rules expressed by this function can be stated as follows:

- All links that are part of a subchain must be free of collisions. Any collision sets the subchain contact quality parameter **subchain_cq** to **MIN_QUALITY**.
- If there are no collisions, the subchain contact quality parameter **subchain_cq** is set to the minimum of the contact quality values of the links that make up the subchain.

The complete set of algorithms using function **link_axis_quality_scan** can be found in Appendix A under top-level function **lower_bound_gq_3_parallel**.

The circuits of Figure 6.6 can be modified to collect subchain and link contact quality information simultaneously as shown in Figure 6.8. At the origin of a routing path, the **subchain_cq** input is set to zero, and the **max_sq** input is set to the quality value of the best subchain reachable from the current processor and link orientation (**max_sq[link_orient, i]** in function **finger_quality_3_p**). Because the desired solution will eventually be the minimum of the **max_sq** input value and the link contact quality value computed for the current link configuration, a **MIN** box is added to the circuit to keep **subchain_cq** at or below the level of the **max_sq** line. This is not really necessary here, as a single **MIN** could be executed at the end of the routing path, but as Section 6.7 will show, it is useful to keep an accurate estimate of the quality value of the *subchain* seen so far rather than just a measure of the contact quality value of the *link* seen so far on the **subchain_cq** line. The LINK-CQ boxes of Figure 6.7 are replaced by the SUBCHAIN-CQ boxes of Figure 6.8 to make use of this new circuit.

A revised complexity estimate is given by:

$$O\left(\left\lceil\frac{R^3}{P}\right\rceil\left((f_E + f_T) + LO^3\left\lceil\frac{d_{link}P^{\frac{1}{3}}}{S}\right\rceil\right)\right). \tag{6.8}$$

Compare this to Expression 6.2. The first half of this expression is required for the one-time computation of the local geometric information. Multiplier LO^3 has been removed from this part of the computation. The second half of the expression represents the time already required to execute the **fetch** commands that pass subchain quality information from a link to its proximal neighbor (see Expression 6.6). During execution of these **fetch** commands, a current link contact quality measure can be reconstructed from the stored, local geometric information.

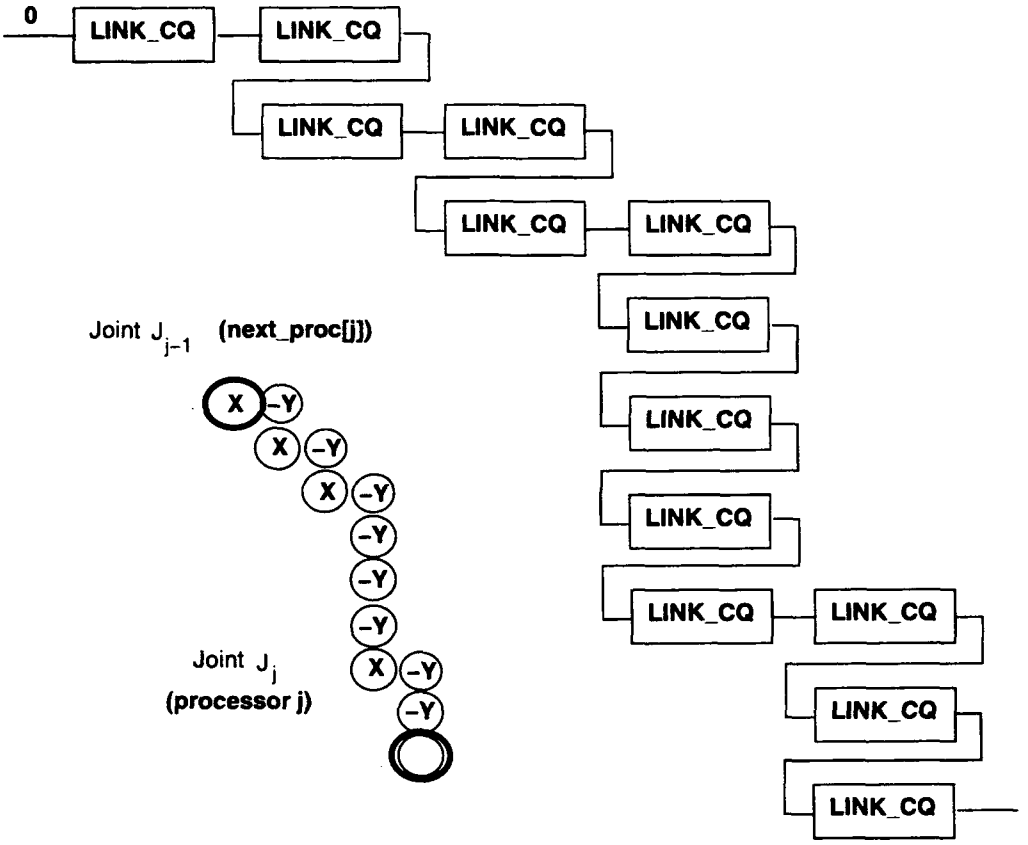


Figure 6.7: Reconstruction of the contact quality measure for a link. This is implemented using a scan through the processor grid, but it is equivalent to chaining together the inputs and outputs of a number of simple circuits. This is true because each individual processor does very little, executing only a set of very simple operations and passing the results to the next processor along the routing path.

```
link_axis_quality_scan
   (max_sq, link, link_orient, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   for i in processors() in parallel do
      subchain_cq[i] := min(0, max_sq[i]);

   for step in routing_path_of(link, link_orient, hand_model) do
   begin
      diam := diam_of(location_of(step), link, hand_model);
      for i in processors() in parallel do
      begin
         local_cq[i] :=
            compute_local_cq
               (diam, collision_dist_of(lc_info[i]), contact_dist_of(lc_info[i]),
                  contact_point_of(lc_info[i]), contact_feature_of(lc_info[i]), link,
                  link_orient, orient, contact_assignment, prototype);
         if local_cq[i] = MIN_QUALITY then
            subchain_cq[i] := MIN_QUALITY;
         if subchain_cq[i] > MIN_QUALITY then
            subchain_cq[i] := min(max(local_cq[i], subchain_cq[i]), max_sq[i]);
      end
      subchain_cq := local_fetch(subchain_cq, direction_of(step));
      max_sq := local_fetch(max_sq, direction_of(step));
   end
   return(subchain_cq);
end
```

Algorithm 6.5:

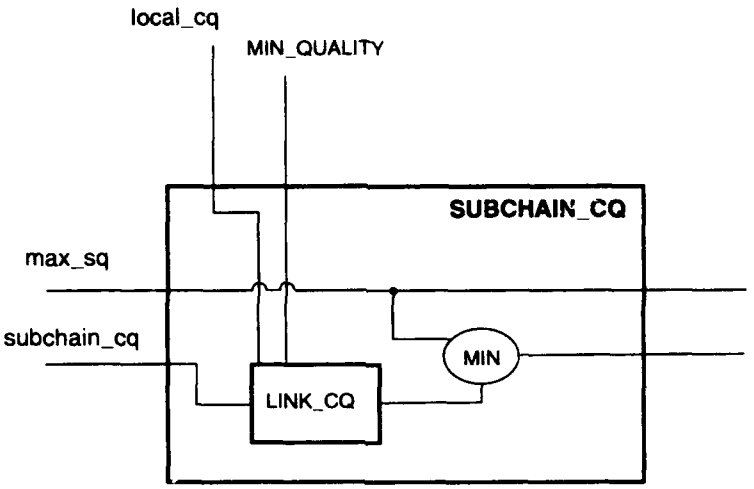


Figure 6.8: This circuit updates the link contact quality measures, here represented with **subchain_cq**, making sure that this value stays at or below the optimal subchain contact quality value **max_sq**.

6.6 Working with Less Memory

Sections 6.4 and 6.5 have shown how the parallel algorithm of Chapter 5 can be made more efficient by keeping processing as local as possible, and thereby reducing expensive communication costs. Section 6.4 showed that subchain quality information can be maximized locally before it is passed from one link to the next. Section 6.5 showed that local geometric information can be distributed over the processor grid representing position space. This information can then be used to reconstruct link contact quality measures, avoiding the need to repeatedly perform geometric calculations dependent on the complexity of the hand, target object, and environment geometries.

This section considers another type of problem. Although a three-dimensional grid of processors has been assumed, six-dimensional arrays of subchain finger and grasp quality values continue to be computed and stored. It is likely that limitations on processor memory will not allow these arrays to be stored at sufficient resolution to reconstruct kinematically accurate hand configurations. This section describes a modified algorithm that allows for good kinematic accuracy and requires very little temporary storage space. Little additional space over what is required to store complete solutions is used. Unfortunately, this algorithm pays in execution speed what it gains in storage space.

A variety of problems arise with the current algorithm if orientation space is sampled very coarsely. One such problem is represented in Figure 6.9. If orientation space is sampled very coarsely, each sample orientation must represent a large range of orientations. At the wrist, optimal fingers may be returned that correspond to opposing extremes of this range. This causes the problem shown in Figure 6.9, where no configuration of the

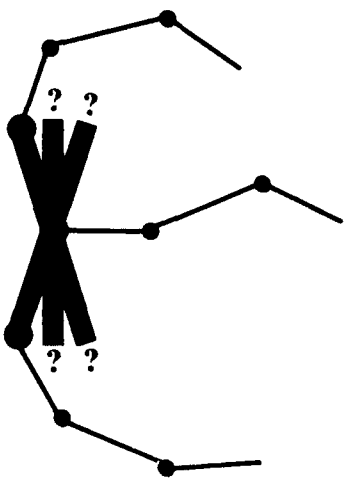


Figure 6.9: The three fingers correspond to different wrist orientations. A hand configuration that will closely fit all three fingers may be difficult to achieve.

rigid wrist segment will fit all three fingers perfectly. This problem can be much worse for a three-dimensional hand, where orientation space has three dimensions. Connections between links may contain large errors that are out of the plane of motion of the joint connecting the two links. When a hand is fit to a set of joint positions containing such errors, it is difficult to compensate with local changes in joint angle, and these errors propagate, affecting the entire hand configuration.

The main source of the problem is the breadth-first nature of the optimization process described in function **finger_quality_3_p** (see Algorithm 6.6 below). Before a link can be added to any optimal subchain, the entire space of subchain qualities must be computed for the distal neighbor of that link (reference point A). The advantage of this process is that it is fast. Optimal subchain quality values are available for reference (B), and similar subchains do not have to be constructed and tested many times. The disadvantage of this process is that it requires a large amount of memory. In order to approximate hand configurations accurately, orientation space would need to be sampled finely.

An algorithm that requires much less storage than the current algorithm can be found in Appendix A under top-level function **lower_bound_gq_4_parallel**. Portions of this algorithm are examined below. This algorithm does not compute and store an entire space of subchain quality values. Instead, it calculates these values as needed. This has the advantage that each link configuration can be tailored exactly to the hand configuration under construction. The hand configurations that are sampled are very accurate and virtually no temporary storage is required. The algorithm has the disadvantage that it can no longer be described as a dynamic programming algorithm, as information is continually being recomputed. In this algorithm, the grasp prototype is only useful for avoiding

computation of the actual grasp quality measure as described in Section 5.6 and for separating the contributions of the fingers of the hand, allowing finger qualities to be optimized independently.

It is useful to examine the differences between the original breadth-first algorithm and the new depth-first algorithm. The first real difference is seen in functions **compute_gq_3_parallel** and **compute_gq_4_parallel**. In both, the complete finger quality array is computed over the space of proximal link configurations of each finger, but in the depth-first algorithm, the sampling of orientation space is pulled out of function **finger_quality_4_p**. In other words, the code sequence:

```
for finger in fingers_of(hand_model) do
begin
   prox_link := prox_link_of(finger, hand_model);
   fq[finger] :=
      finger_quality_3_p
         (prox_link, orient, contact_assignment, prototype, hand_model,
            local_contact_info);
end
```

in function **compute_gq_3_parallel** becomes

```
for finger in fingers_of(hand_model) do
begin
   prox_link := prox_link_of(finger, hand_model);
   for wrist_orient in workspace_orients() do
   begin
      prox_orient := prox_orient_of(finger, hand_model, wrist_orient);
      fq[finger, wrist_orient] :=
         finger_quality_4_p
            (prox_link, prox_orient, orient, contact_assignment, prototype,
               hand_model, local_contact_info);
   end
end
```

in function **compute_gq_4_parallel**.

Recursive function **finger_quality_4_p** has been redefined to return not an entire configuration space of subchain quality values, but the space of subchain quality values having proximal links in a given orientation. This reduces the subchain quality array returned by this function from a six-dimensional array over configurations to a three-dimensional array over positions.

```
finger_quality_3_p (link, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   {find maximal subchain quality}
   if no_next_link?(link, hand_model) then
      {subchain quality can be set to MAX_QUALITY}
      for link_orient in workspace_orients() do
         for i in processors() in parallel do
            max_sq[link_orient, i] := MAX_QUALITY;
   else
   begin
      {get subchain_quality for next link, all link orients}
      next_link := next_link_of(link, hand_model);
A     subchain_quality :=
         finger_quality_3_p
            (next_link, orient, contact_assignment, prototype, hand_model, lc_info);

      {get subchain_quality for current link, all link orients}
      for link_orient in workspace_orients() do
         for i in processors() in parallel do
         begin
            {maximize subchain_quality over joint angles}
            max_sq[link_orient, i] := MIN_QUALITY;
            for theta in joint_angles_of(link, hand_model) do
            begin
               next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
B              if subchain_quality[next_orient, i] > max_sq[link_orient, i] then
                  max_sq[link_orient, i] := subchain_quality[next_orient, i];
            end
         end
   end

   {collect link info along link axis, merge with subchain info for all link_orients}
   for link_orient in workspace_orients() do
      fq[link_orient] :=
         link_axis_quality_scan
            (max_sq[link_orient], link, link_orient, orient, contact_assignment,
               prototype, hand_model, lc_info);
   return(fq);
end
```

Algorithm 6.6:

```
finger_quality_4_p
   (link, link_orient, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   {find maximal subchain quality}
   if no_next_link?(link, hand_model) then
      {subchain quality can be set to MAX_QUALITY}
      for i in processors() in parallel do
         max_sq[i] := MAX_QUALITY;
   else
   begin
      {maximize subchain_quality over joint angles}
      next_link := next_link_of(link, hand_model);
      for i in processors() in parallel do
         max_sq[i] := MIN_QUALITY;
      for theta in joint_angles_of(link, hand_model) do
      begin
         {get subchain_quality for current link and link_orient}
         next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
C        subchain_quality :=
            finger_quality_4_p (next_link, next_orient, orient, contact_assignment,
                                prototype, hand_model, lc_info);
         for i in processors() in parallel do
            if subchain_quality[i] > max_sq[i] then
               max_sq[i] := subchain_quality[i];
      end
   end

   {collect link info along link axis, merge with subchain info}
   fq := link_axis_quality_scan
            (max_sq, link, link_orient, orient, contact_assignment, prototype,
               hand_model, lc_info);
   return(fq);
end
```

Algorithm 6.7:

Other differences can be seen between functions **finger_quality_3_p** and **finger_quality_4_p**. (The complete functions are shown in Algorithms 6.6 and 6.7.) The breadth-first function in Algorithm 6.6 first constructs a complete array of subchain quality values over the six-dimensional space of configurations of the distal neighbor of input parameter **link** (reference point A). The body of the breadth-first function consists of accessing this space to fill another six-dimensional space that extends the optimal subchains by a single link (B). In the depth-first function of Algorithm 6.7, computation of subchain quality is not the first step. This computation is embedded within the loop over joint angles **theta** (C). This results in the transformation from a breadth-first to a depth-first algorithm. The recursive call to **finger_quality_4_p** generates from scratch optimal subchains terminating in orientation **next_orient** of the distal neighbor of link (C). This allows the kinematics of these subchains to be made very accurate. Unlike before, this function does not simply extract from a precomputed array the quality measure found in the orientation space slot that is closest to **next_orient**; instead, it reconstructs the optimal subchain quality measure for the range of subchains terminating exactly in orientation **next_orient** within the limitations imposed by the sampling of position space. Embedding the recursive call within the joint angle loop also eliminates the space required to store the subchain quality array, because this array does not have to be precomputed. This algorithm will be slower, however, as similar subchain quality values cannot be reused.

A time estimate for the depth-first algorithm is given by:

$$O\left(FO^3\left\lceil\frac{R^3}{P}\right\rceil(O'd_{angle})^{\frac{L}{F}}\left(O'd_{angle}+\left\lceil\frac{P^{\frac{1}{3}}d_{link}}{S}\right\rceil\right)\right). \tag{6.9}$$

The expression contains an outer loop over all fingers F and all wrist orientations O^3. Position space is sampled in parallel using $\left\lceil\frac{R^3}{P}\right\rceil$ virtual processors. The number of links that must be reconstructed to form an optimal finger quality measure depends exponentially on the number of links in a finger $\left(\frac{L}{F}\right)$, and the branching factor of the tree representing sampled finger configurations can be approximated with $O'd_{angle}$, where joint angle sampling density O' is now distinct from parameter O^3, used to describe the number of orientation space samples in the solution. With this algorithm, only a small number of solutions in which hand configurations are sampled very finely need be stored. That is, parameter O^3 could be relatively small and O' could be relatively large. Each link contact quality measure must be reconstructed by pushing information through the processor grid from one joint to the next. Time to perform this operation can be approximated as $\left\lceil\frac{P^{\frac{1}{3}}d_{link}}{S}\right\rceil$.

Compare this expression to Expression 6.7. The difference between these two expressions describes the tradeoff between the breadth-first and depth-first algorithms. Most importantly, while the breadth-first algorithm depends linearly on the number of links in the hand L, the depth-first algorithm depends linearly on the number of fingers F and exponentially on the number of links per finger $\frac{L}{F}$.

The memory tradeoff has also been outlined. The breadth-first algorithm requires that six-dimensional subchain quality arrays be stored, while the depth-first algorithm requires

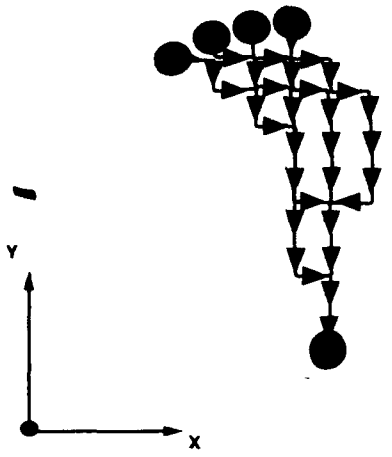


Figure 6.10: Routing information along a link axis for a range of link orientations results in a routing tree, not a unique routing path.

that three-dimensional subchain quality arrays be stored. A quantitative comparison of these algorithms for one implementation on a specific parallel computer is given in Section 7.2.

6.7 Working with Fewer Samples

The previous section described a technique for overcoming a particular problem: the problem of large memory requirements. Unfortunately, the memory savings obtained using this technique does not come for free, and the cost is paid in time required to execute the new set of algorithms. There are few options left for keeping time requirements low, and the most obvious of these remaining options is to limit the number of samples taken of orientation space. In Expression 6.9, this would be to place a limit on both parameter O^3, which represents the number of wrist orientations for which a grasp quality estimate is computed, and parameter O' in expression $O'd_{angle}$, which determines the number of joint angle samples explored at each joint. The problem with greatly limiting these orientation parameters is that coverage of hand configuration space is lost. If only a widely distributed (but accurate) space of hand configurations is examined, then important solutions may be missed. This section describes one technique that can be used to help overcome this problem.

Solutions can be missed using the algorithms of Section 6.5 and Section 6.6, because

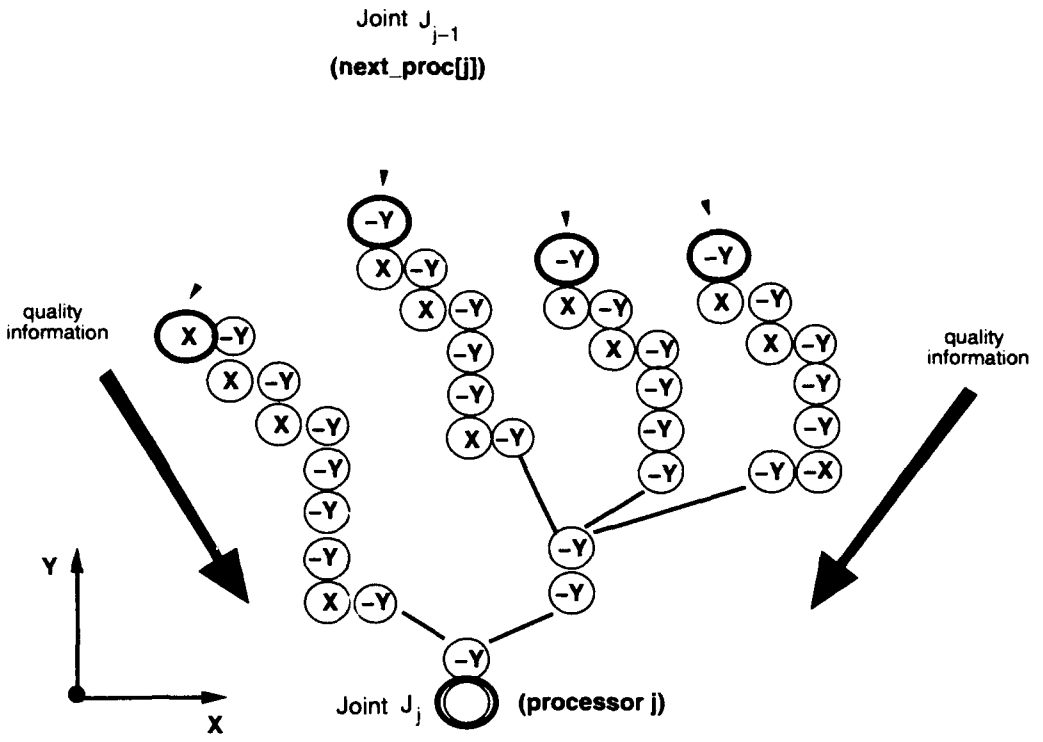


Figure 6.11: A tree to direct the routing of subchain quality information along a link axis for a range of link orientations.

they assume that each configuration space sample represents a single point in configuration space. In particular, they assume that from the position and orientation of a link and from a joint angle, the position and orientation of the distal neighbor of that link can uniquely be specified. This assumption is represented by the fact that the routing path of Figures 6.2 and 6.3 is a single path.

If orientation space is to be sampled coarsely, better coverage of configuration space will be obtained by using a *routing tree*, as shown in Figures 6.10 and 6.11, rather than a single routing path. This means acknowledging that the problem of finding the configuration of the distal neighbor of a link given that link's configuration and a joint angle may result in a range of solutions, not a unique solution. The next few sections show how routing trees can be used rather than routing paths to reconstruct link contact quality measures and retrieve optimal subchain quality measures.

6.7.1 Computing a Link Quality Measure

In Section 6.5.2, the rules for using a routing path to obtain a link contact quality measure were stated as follows:

- All processors representing points on the link axis must indicate that the link lies in free space at that point. Any collision sets the link contact quality parameter **link_cq** to **MIN_QUALITY**.

- If there are no collisions, the link contact quality parameter **link_cq** is set to the maximum of all local contact quality values **local_cq[i]**.

To expand these rules for use with a routing tree, *some* path must be found where the link lies entirely in free space. The contact quality measure over all such paths is maximized. This new set of rules can be restated as follows:

- All processors representing points on the link axis *of some path* must indicate that the link lies in free space at that point. Any collision *along a path* sets the link contact quality parameter **link_cq** *of that path* to **MIN_QUALITY**.

- If there are no collisions *along a path*, the link contact quality parameter **link_cq** *of that path* is set to the maximum of all local contact quality values **local_cq[i]**.

- The **link_cq** value for the routing tree is taken to be the maximum **link_cq** value for any path in the routing tree.

Note that the contact quality measure for each path through the tree does not need to be separately computed. Contact quality measures can be combined where any set of paths merges. This can be described very easily by adding to the LINK-CQ circuit of Figure 6.6 some preprocessing to merge two incoming paths (Figure 6.12). The function executed at each processor is the same except when two paths merge. Here the maximum incoming **link_cq** value is retained.

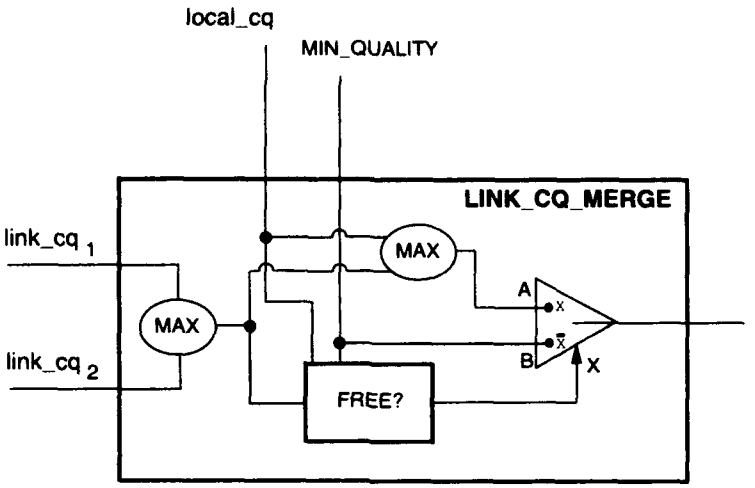


Figure 6.12: When merging two paths through a routing tree designed for reconstructing a link contact quality measure, it is sufficient to begir with the maximum **link_cq** value for the two paths.

Figure 6.13 gives an example of the merging process just described. A link contact quality measure is computed by optimizing this measure over the range of link configurations shown. The range of link configurations maps to a tree of paths through the Cartesian space processor grid. Free space and contact information are passed from the leaves of this tree to the root, the proximal end of the link axis.

Figure 6.14 shows the example, along with a target object and an obstacle. Taking the diameter of the link into account, it can be seen that some paths through the grid will register collisions, and some paths will register contact.

Figure 6.15 shows an abbreviated representation of the merging process for this example. The tree of paths represents the proximal seven nodes of the routing tree for the problem shown in Figure 6.14. Symbols X, Y, and $-Y$ indicate the direction in which information is passed when moving from node to node up the tree. Identical nodes are indicated by squiggly lines, but paths cannot be merged at these nodes because they diverge both above and below the coincident points. Finally, collisions are indicated with shaded boxes, and contact is indicated with a number giving the local quality value of the contact. One safe path through the tree exists.

Figure 6.16 shows a step by step illustration of how information is passed up the tree. Collisions are propagated upward, unless *some* free path can be found. When there is a free path, the maximum quality value is stored. The result of this scan shows that there is a path that is free of collisions and in contact with the target object, with quality 0.3.

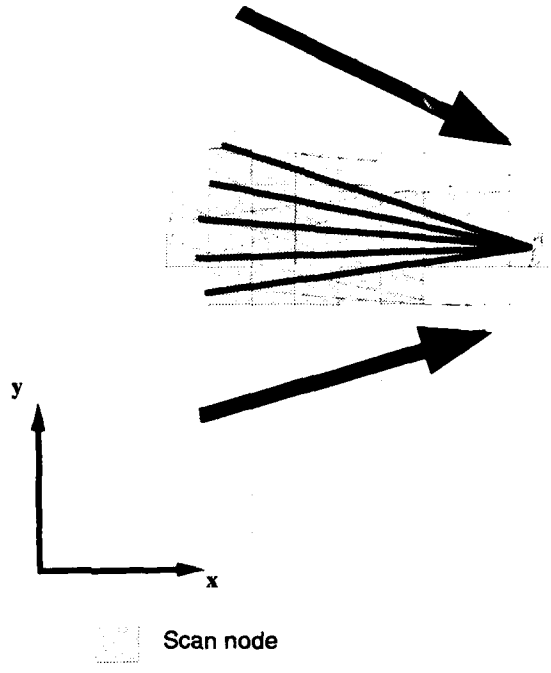


Figure 6.13: A scan to collect collision and contact quality information along a range of link orientations.

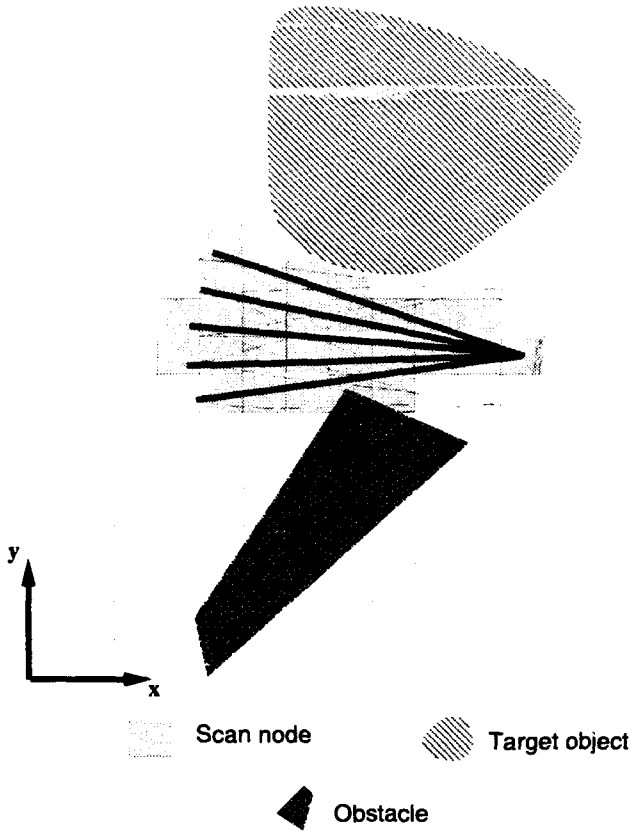


Figure 6.14: A scan to collect collision and contact quality information along a range of link orientations. An obstacle and the target object have been added to the figure. Some link orientations in this range are in collision with the obstacle. Some are in contact with the target object.

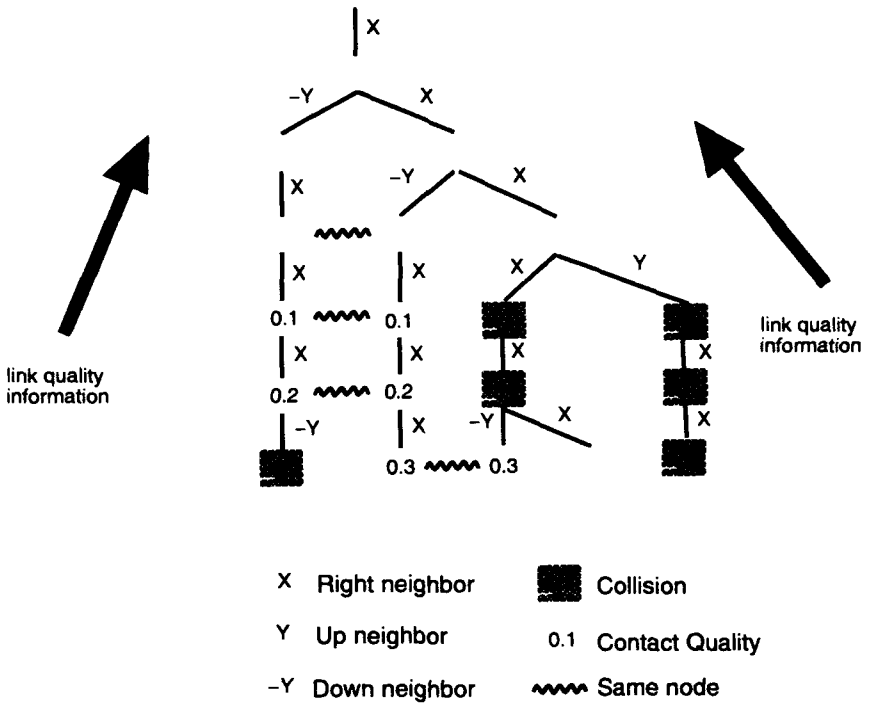


Figure 6.15: A routing tree, indicating local collision conditions and contact quality values.

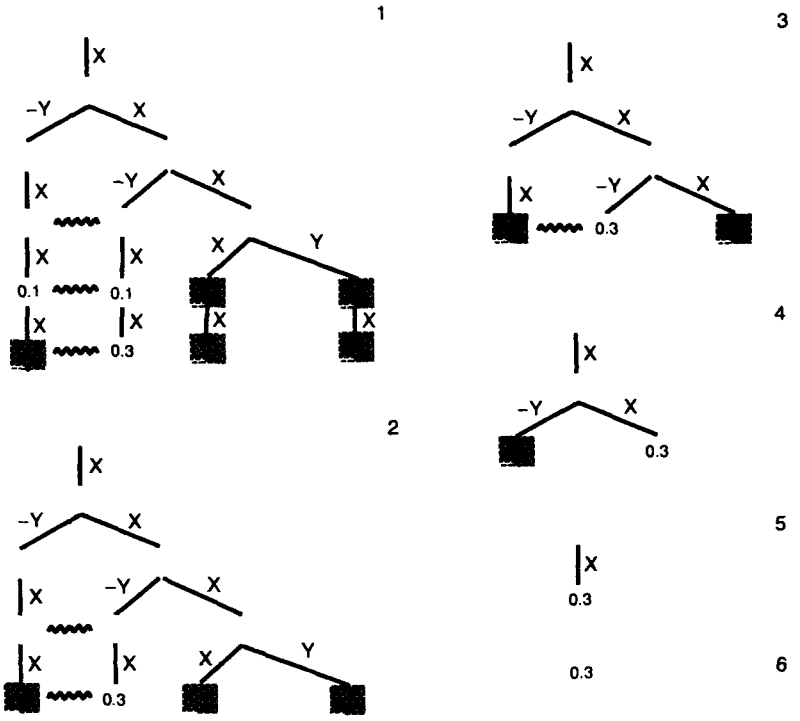


Figure 6.16: A step by step illustration of the scanning process. This tracks optimal **link_cq** values as information is passed from the proximal to the distal end of the link.

6.7.2 Computing a Finger Quality Measure

Section 6.5 noted that subchain contact quality values could be passed up a link axis while the link contact quality measure was constructed. This was more efficient than performing two nearly identical **fetch** operations, and the effect of including the subchain contact quality value was very simple, as illustrated in the circuit of Figure 6.8. The **subchain_cq** value was simply kept below the **max_sq** value at all times.

This approach can also be used when there is a routing tree instead of a single routing path, but the situation is somewhat more complicated. In particular, the value on the **max_sq** line must always reflect the best **max_sq** value reachable along any path *free of collisions.*

The pair of circuits shown in Figure 6.17 illustrates the function required to merge two paths in the routing tree. A set of SUBCHAIN-CQ boxes from Figure 6.8 and SUBCHAIN-CQ-MERGE boxes from Figure 6.17 can be strung together as determined by the structure of a routing tree to extend the subchain contact quality value to a subchain one additional link in length.

In a pseudocode version, function *link_axis_quality_scan* (Algorithm 6.5) would have a routing tree in place of a routing path. This tree could, for example, be traversed depth-first. The pseudocode version is not shown here.

6.8 Summary and Discussion

This chapter described three problems encountered when implementing the parallel algorithm of Chapter 5:

1. **Too few processors:** The number of processors available does not allow each processor to be assigned to a small chunk of configuration space.

2. **Too little memory:** The amount of on-processor memory limits the accuracy that can be obtained using the current, breadth-first algorithm.

3. **Too little time:** The amount of time available for executing this algorithm limits the density at which hand configuration space can be sampled. If this sampling is coarse, but accurate, good solutions may be missed.

The first problem was addressed by mapping the three-dimensional space of *positions* of a link or of the robot wrist onto a three-dimensional grid of processors rather than attempting to assign a unique processor to each six-dimensional link or wrist configuration. This mapping allowed a particular implementation of the **fetch** function to be defined, and the algorithm was optimized for this implementation. It also allowed a distributed representation of target object and environment geometries to be used, avoiding the repetition of expensive collision and contact detection operations.

The second problem was addressed by proposing an alternative depth-first algorithm that consumes much less storage space, but requires considerably more execution time.

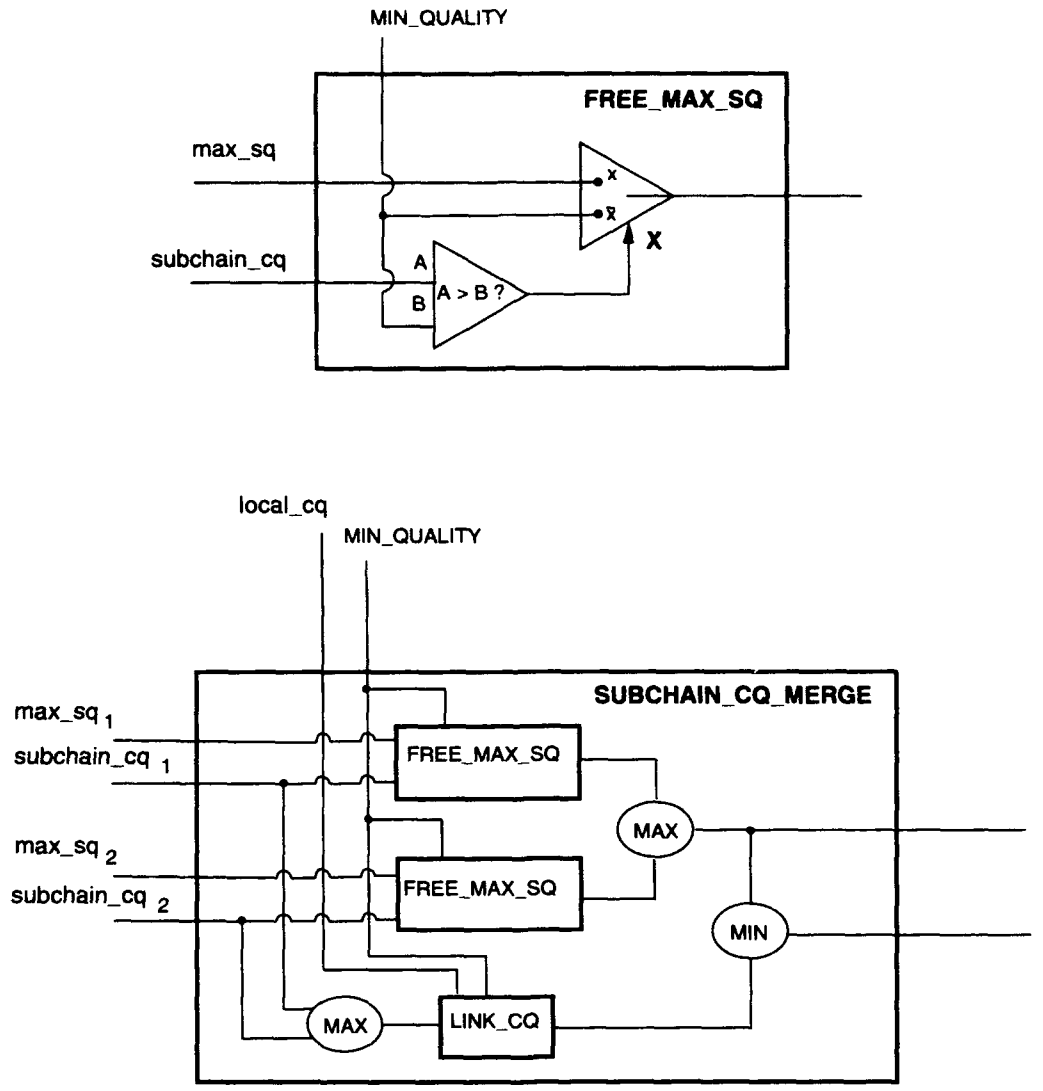


Figure 6.17: Circuits to merge two paths during computation of optimal subchain quality measures. The **subchain_cq** line must be kept below the quality measure of the best *free* subchain (the best **max_sq** value for a free subchain.)

The third problem was addressed by proposing the use of routing trees to allow for complete coverage of hand configuration space even when the sampling of this configuration space is very coarse. A continuous tradeoff can be made between sampling accuracy and execution time.

Although order notation is used to compare the various algorithms throughout the text of this chapter and Chapter 5, it is helpful to plug in one set of values for the parameters in these expressions to obtain a quantitative comparison of the algorithms. Figure 6.18 shows one list of parameter values similar to those that will be used in Chapter 7. It also shows a breakdown of four of the algorithms into the following operations:

- **Lk-Env Coll:** Determine whether there is a collision between a link of the hand and any object or obstacle in the environment.
- **Pt-Env Coll:** Find the distance from a point to the nearest object or obstacle.
- **Vect Prod:** Compute a six-dimensional dot product or cross product.
- **Float Comp:** Decide if one floating point number is greater than, less than, equal to another.
- **Fetch:** Each processor gets some (small) amount of information from another processor.
- **Local Fetch:** All processors get some (small) amount of information from a neighboring processor. The neighboring processor is one of its six nearest neighbors in the three-dimensional grid, and the direction of information transfer (i.e. up, down, east, west, north, south) is the same for all processors.

The values in the table represent the number of times each operation is repeated during construction of the grasp quality space for a given orientation of the grasp prototype with respect to the target object and for a given assignment of a link of the hand to a contact of the grasp (i.e. for a given value of parameters **orient** and **contact_assignment** in the algorithms). For the parallel algorithms, these are the number of operations required *for each physical processor.* The amount of memory required (per processor) for each algorithm is estimated in the far right-hand column. It is assumed that none of these algorithms use routing trees.

The first algorithm in Figure 6.18, **lower_bound_gq_1**, is the original serial algorithm for computing a grasp quality space using a grasp prototype. The second algorithm, **lower_bound_gq_1_parallel**, is the original parallel version of this algorithm, presented in Chapter 5. Because there are assumed to be 8000 physical processors, the second algorithm performs approximately a factor of 8000 fewer operations in each physical processor. Some communication is required, but it will probably have a small effect.

The third algorithm in Figure 6.18, **lower_bound_gq_3_parallel**, represents the parallel algorithm after the three-dimensional Cartesian space of link or wrist positions has been mapped onto the processor grid and the breadth-first algorithm has been optimized for this mapping, as described in Sections 6.4 and 6.5. The most obvious change from function

$$
\begin{aligned}
O^3 &= 180 \\
R^3 &= 32,000 \\
P &= 8,000 \\
L &= 12 \\
F &= 3 \\
\frac{Od_{angle}}{2\pi} &= 3 \\
\frac{O'd_{angle}}{2\pi} &= 3 \\
\frac{P^{\frac{1}{3}}d_{link}}{S} &= 5
\end{aligned}
$$

Algorithm	Lk-Env Coll	Pt-Env Coll	Vect Prod	Float Comp	Fetch	Local Fetch	Memory
lower_bound_gq_1	70M	0	770M	1.1B	0	0	370MB
lower_bound_gq_1_parallel	8.6K	0	95K	140K	28K	0	46KB
lower_bound_gq_3_parallel	0	4	470K	640K	0	32K	46KB
lower_bound_gq_4_parallel	0	4	3.3M	5B	0	162K	23KB

Figure 6.18: A quantitative comparison of some of the algorithms presented.

lower_bound_gq_1_parallel is that the collision and contact detection operations have been converted into a number of vector operations. The tradeoff seems to be a good one. This algorithm will be at least as fast as the smartest collision-detection algorithms that do not use a distributed representation of the target object and environment geometry, and it is much simpler. In addition, because the mapping of configuration space onto the processor grid is known, the general **fetch** commands have been translated into **local_fetch** commands.

The fourth algorithm, **lower_bound_gq_4_parallel**, represents the depth-first version of the third algorithm. Here each tree of finger configurations is expanded to allow for greater accuracy in sampling. From the numbers in Figure 6.18, it seems that this algorithm will be approximately 8 times slower than the breadth-first version. Note that the depth-first algorithm requires approximately half the memory of the others for this set of parameters.

It is not clear just from looking at the example in Figure 6.18 whether the breadth-first algorithm or the depth-first algorithm would be more effective. With the parameters given, there is little difference in memory requirements (a factor of two). Although the breadth-first algorithm will be significantly faster, it may not be sufficiently accurate. In addition, the entries in the table do not reflect the use of routing trees as described in Section 6.7. Given the sparse sampling of orientation space that was chosen to generate the entries in the table, routing trees will be needed to ensure that good solutions are not missed.

In general, the tradeoffs proposed in Sections 6.6 and 6.7 allow the algorithm to be tuned to balance the following parameters for a particular parallel hardware configuration:

- Accuracy: are the optimal solutions that are constructed kinematically accurate?
- Time: how long does it take to build the grasp quality space?
- Space: how much storage space is required for the temporary quality variables and for storage of the solution array?

The breadth-first algorithm is relatively fast, but it requires large amounts of memory to obtain an accurate sampling of hand configuration space. The algorithm of Section 6.6 is slow, but it samples hand configuration space very accurately, and it requires little more space than that required to store a solution. The use of routing trees as outlined in Section 6.7 allows time and space to be gained at the expense of accuracy.

Chapter 7 addresses the problem of tuning the algorithm, using some quantitative results obtained using a particular parallel machine. These results are used to select between the breadth-first and depth-first algorithms and to choose the set of sampling parameters that are employed in grasp synthesis examples in that chapter.

Chapter 7

Examples Using 3D Objects

Chapter 5 described a parallel algorithm for grasp synthesis, and Chapter 6 reviewed a number of improvements to this algorithm, as well as a number of modifications that could be used to make tradeoffs between execution time, accuracy, and memory requirements. This chapter describes the hardware that was used to test the grasp synthesis algorithm, and it describes a set of experiments that were performed to tune this algorithm for the given hardware configuration. The tuned algorithm was employed to compute spaces of grasp quality measures for a variety of objects using a cylindrical grasp prototype, similar to the grasp shown in Figure 7.1. These results are presented and analyzed below.

The breakdown of sections in this chapter can be described as follows. Section 7.1 covers the hardware setup that was used to implement the grasp synthesis algorithm of this report. Section 7.2 describes a set of experiments designed to select between the algorithms described in Chapter 6 and tune them for the given hardware configuration. Section 7.3 covers the problem of friction, which is necessary for the cylindrical examples that will be shown. The cylinder example grasp relies on friction to prevent slipping in two of the six wrench space dimensions. Section 7.4 shows a variety of examples, which illustrate an ability to find grasps of objects more geometrically complex than the prototype object and to find grasps of objects in cluttered environments. Some of these grasps are verified using the Salisbury hand, and photos of these grasps are shown. Section 7.5 presents a summary.

7.1 Hardware

The parallel grasp synthesis algorithm was run on a Thinking Machines Connection Machine CM2. This is a Single Instruction Multiple Data (SIMD) machine, which means that all processors simultaneously execute the same instruction on their local data set. Instructions are broadcast from a front-end machine (here a Sun-4). The particular CM2 used in these experiments has 8000 processors, each with 64K bytes of memory. The machine does have floating point hardware, but every 32 processors share a floating point accelerator. The machine can be configured for fast local communication when all processors

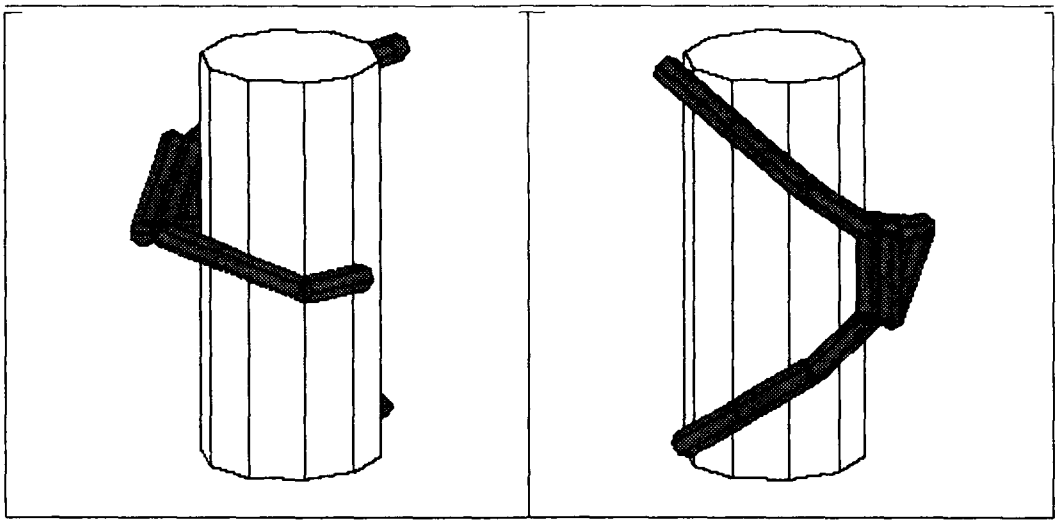


Figure 7.1: A prototype grasp of a cylinder.

send messages in the same direction.

Grasps were tested using the Salisbury hand [42], a three-fingered hand with three joints per finger.

7.2 Depth-first vs. Breadth-first Grasp Optimization

The previous chapter described a set of tools that allowed tradeoffs to be made when implementing an algorithm to compute an array of grasp quality estimates over the space of wrist configurations. The tradeoffs were to be made between:

- Time required to compute an array of grasp quality estimates.
- Space required to store the solution and to store the temporary quality values.
- Kinematic accuracy of the hand configurations associated with the grasp quality estimates of the solution space.

That chapter described a breadth-first algorithm that was relatively fast, but required large amounts of storage space (**lower_bound_gq_3_parallel**). It described a depth-first algorithm that required much less storage and was capable of achieving greater accuracy, but this algorithm was relatively slow (**lower_bound_gq_4_parallel**). It also described a technique (routing trees) that could be applied to either algorithm to increase speed and save space at the cost of accuracy (Section 6.7).

This section describes a pair of experiments performed using the hardware described in Section 7.1. The experiments were designed to quantify, for this hardware setup, the

tradeoffs associated with the tools of the previous chapter. The goal was to select a complete algorithm to meet the following specifications:

1. **Kinematic accuracy is acceptable.** It will almost always be possible to achieve a good fit of the robot hand to an optimal hand configuration returned by the algorithm.

2. **Coverage of hand configuration space is nearly complete.** Good solutions in general will not be missed.

3. **The algorithm is as fast as possible given the first two constraints.**

7.2.1 Experiment 1: Tuning the Breadth-first Algorithm

Because the main objective for this algorithm is really that it be as fast as possible given constraints on kinematic accuracy and hand configuration space coverage, the faster breadth-first algorithm was tried first. This algorithm constructs each space of optimal subchain quality measures by first constructing the complete space of quality measures for the distal link of the chain and then iteratively extending this space for each link of the subchain. It is fast because optimal subchain information is stored and can be reused, but it is only as accurate as the sampling of configuration space used to compute this information allows.

Representing Configuration Space

The first decision made was in how to represent configuration space. Position space was sampled using a 32x32x32 grid. This created 32K grid elements, each representing a cube 0.5″ on a side. The 32K grid elements were distributed over the 8K processors of the machine so that each physical processor simulated 4 virtual processors.

Each physical processor has 64K bytes of memory, so 16K bytes of memory was available for each virtual processor. Each processor needed space to store optimal solutions and grasp quality estimates for 2 complete orientation maps. A slot in an orientation map had to be large enough to store approximately 256 bits of information. Some working storage was also required for performing a variety of vector operations. The maximum size orientation map that could be accommodated given these constraints was found to be a map of 180 orientations.

Tuning the Routing Trees for Complete Configuration Space Coverage

An orientation map having only 180 orientations made it very important that slots in these orientation maps be represented as ranges and not points in orientation space. In other words, it was necessary to use routing trees rather than routing paths as described in Section 6.7. The next decision that had to be made was in how broad to make the routing trees.

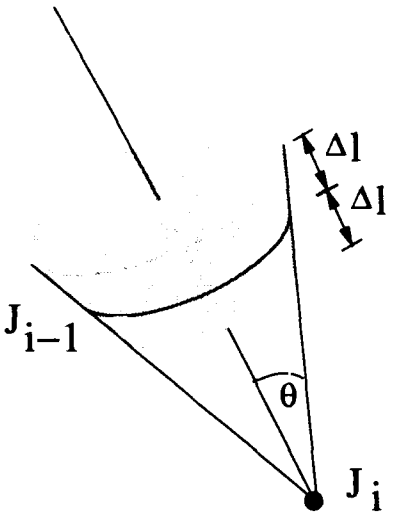


Figure 7.2: A range of link configurations can be represented using a cone. The link can be rotated about the joint J_i no more than an angle of θ from the nominal orientation. The length of the link is allowed to have error of $\pm\Delta l$.

Each routing tree was designed to collect subchain quality information for a range of link orientations. When embedded in Cartesian space, the routing trees are seen to collect information from a circular portion of the surface of a sphere (Figure 7.2). Links with their proximal joint (joint J_i in Figure 7.2) fixed at a point and having the nominal link orientation indicated by the central axis from joint J_i to joint J_{i-1} can, because of uncertainty in the actual link position and orientation, have their distal joint in any part of the shaded volume shown in Figure 7.2. The parameters associated with defining the portion of space over which subchain quality information is collected are:

- The maximum acceptable variation in link orientation, represented in the cone angle (θ in Figure 7.2).
- The maximum acceptable variation in link length, represented in the thickness of the sphere surface (Δl in Figure 7.2).

The second stage of the experiment consisted of tuning these parameters to achieve fairly complete configuration space coverage. Configuration space coverage was tested in the following way. Given a particular wrist configuration and particular finger, the fingertip grid points that should be reachable in an exact representation of finger configuration space were tested against the fingertip grid points reachable using the breadth-first algorithm and a particular set of routing trees. The parameters of the routing trees were adjusted until the region of grid points reachable by the fingertip was seen to be continuous. Variation in link length was calibrated with respect to variation in link orientation by

attempting to make the region of reachable fingertip points resemble as much as possible the accurate region of reachable fingertip points. This resulted in a cone angle of 0.4 radians and a link length tolerance of 0.3 inches. (A typical joint angle range is π radians, and a typical link length is 1.5 inches.)

Results from Experiment 1

The results of the experiment are shown in Figure 7.3. This figure shows two representations of the position space grid, one on the left, another on the right. Each small box represents a horizontal slice taken along the xy plane of this position space, and the boxes are read top to bottom, left to right up the z axis of this space. The left figure shows the fingertip points that should be reachable by a given finger based at a given wrist orientation. The right figure shows the fingertip points reachable by the tuned version of the breadth-first algorithm. It is clear that this reachable fingertip region is not very accurate. Running this test took 15 seconds. The **fetch** command required to pull information up a routing tree from one joint to the next was called 1080 times.

7.2.2 Experiment 2: Tuning the Depth-first Algorithm

Because the first experiment produced results that were not very accurate, a second experiment was run to test the depth-first algorithm. This algorithm does not construct complete intermediate subchain quality spaces. Subchain quality information is instead constructed as needed. This algorithm should be more accurate than the breadth-first algorithm of experiment 1 because subchains can be accurately constructed. It will be slower because subchain quality information is not stored and cannot be reused. The hope in performing this second experiment was that greater accuracy could be achieved with a minimal cost in execution time and no increase in storage space.

Representing Configuration Space

Configuration space was represented much as in Experiment 1. Position space was sampled using a 32x32x32 grid, set up using 4 virtual processors per physical processor. A solution space of 180 wrist orientations was stored, and the number of joint angle samples taken for each joint of the hand was set to be approximately the same as in Experiment 1. Only 3 joint angle samples were tested for a typical joint angle range of π.

Tuning the Routing Trees for Complete Configuration Space Coverage

Configuration space coverage was tested exactly as in Experiment 1. A wrist configuration and a finger were specified, and the routing tree parameters were tuned to obtain a continuous region of reachable fingertip points that looked as much like the exact region of reachable fingertip points as possible. The routing parameters of Experiment 1—a cone angle of 0.4 radians and a link length tolerance of 0.3 inches—were also found to work well in this experiment.

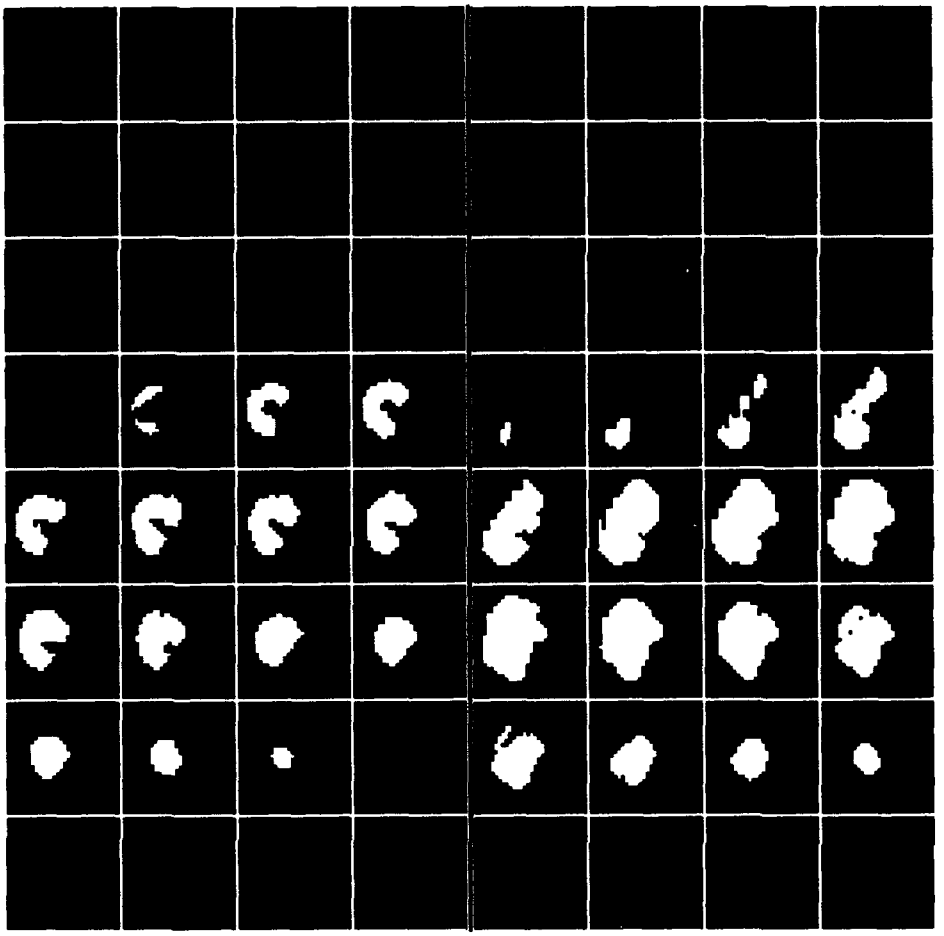


Figure 7.3: Tip positions reachable by a single finger at a single wrist orientation are shown in these figures. The figures are read left to right, top to bottom, and show 32 slices perpendicular to the z-axis. The left picture shows the tip points that should be reached; the right picture shows those that are reached by a breadth-first sampling scheme in a 180 element orientation space.

Results from Experiment 2

The results of this experiment are shown in Figure 7.4. The left figure shows the fingertip points that should be reachable by a given finger based at a given wrist orientation. The right figure shows the fingertip points reachable by the tuned version of the depth-first algorithm. By comparing this figure to Figure 7.3, it is clear that the depth-first algorithm has substantially better accuracy. Running this test took 90 seconds, 6 times longer than in Experiment 1. The **fetch** command required to pull information from one joint to the next was called 30,780 times, a factor of 28.5 more calls than were required for Experiment 1. This factor of 28.5 is not reflected in the execution time because the breadth-first algorithm of Experiment 1 relies on intermediate subchain quality maps and it must spend a substantial amount of time accessing local memory to read these maps. The depth-first algorithm of Experiment 2 has no such maps. Virtually all the execution time of this algorithm is spent running the **fetch** command.

7.2.3 Summary and Tool Selection

Experiment 1 was designed to determine whether the breadth-first algorithm could be used. Memory limitations restricted the orientation space map to 180 slots, and the resulting hand configurations were found to have insufficient accuracy. Experiment 2 was designed to test the more accurate, but slower depth-first algorithm. Wrist orientation space sampling and joint angle sampling in Experiment 2 were set to be approximately the same as that in Experiment 1, not because of memory limitations (especially in the case of joint angle sampling), but in order to keep execution time as low as possible. Much more accurate results were obtained at a cost in execution speed of a factor of 6.

The examples below all use:

- the depth-first algorithm (**lower_bound_gq_4_parallel**)
- with routing trees collecting information from
  - a cone of angle 0.4 radians (θ in Figure 7.2), and
  - a link length tolerance of ± 0.3 inches (Δl in Figure 7.2).

7.3 Friction

Throughout this report, frictionless grasps have been assumed. This is a reasonable assumption, because a high-quality frictionless grasp will only have a better grasp quality measure when friction is considered. The grasp prototype in this chapter is a cylinder grasp, however (Figure 7.1). This grasp prototype is designed to rely on friction. If the target object was an ideal frictionless cylinder, it would slide right out of the grasp when the robot tried to lift it. It would also have no resistance to torque about the vertical cylinder axis. To estimate the quality of this grasp, friction must be considered, and this section describes how this can be done.

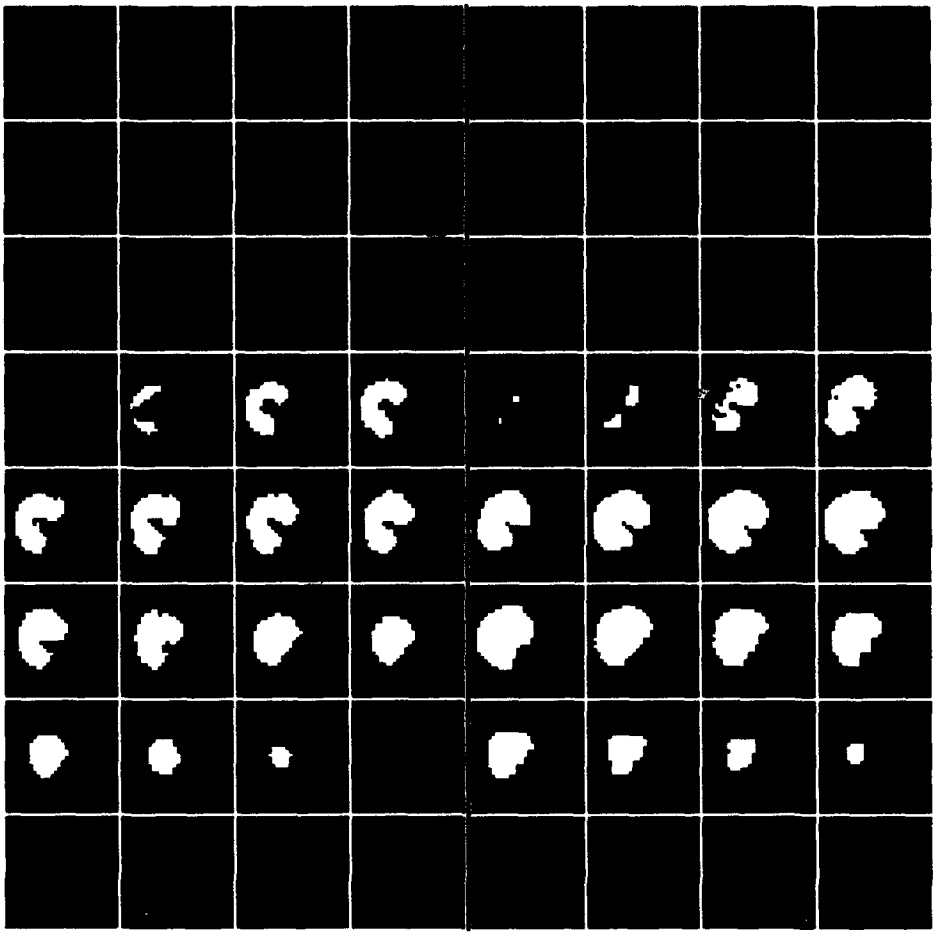


Figure 7.4: Tip positions reachable by a single finger at a single wrist orientation are shown in these figures. The figures are read left to right, top to bottom, and show 32 slices perpendicular to the z-axis. The left picture shows the tip points that should be reached; the right picture shows those that are reached by the depth-first sampling scheme used here.

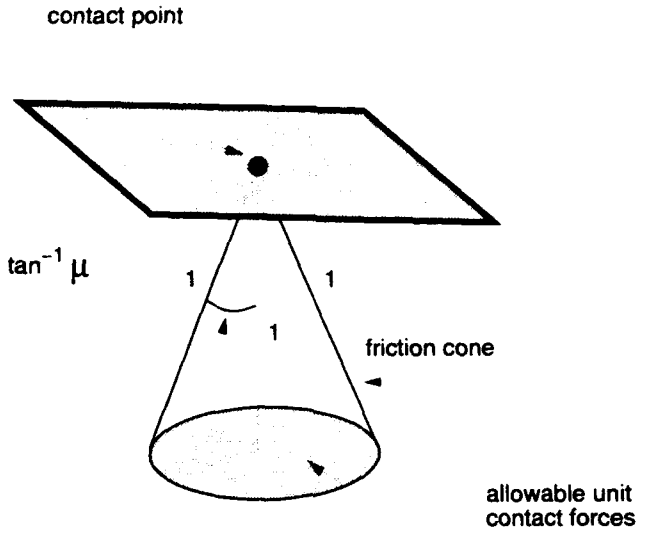


Figure 7.5: A cone representing the space of legal unit forces at a contact with friction. The size of the friction cone is limited by the coefficient of friction μ.

One way to generalize and apply a grasp prototype with friction is to fully incorporate friction into the solution process, so that grasp quality measures can be computed in the full six-dimensional wrench space. This approach was tried, but it has some problems. The presence of friction means that there is a range of contact forces that can be applied at each contact (Figure 7.5). The technique used to generalize grasps in Chapter 3, however, relies on a grasp described using a discrete number of contact wrenches. The space of contact wrenches corresponding to each contact with friction must be sampled to obtain a set of discrete contact wrenches (Figure 7.6). If there are m samples per contact, a c-contact grasp would be transformed into an (mc)-contact grasp.

Now, to match a new grasp to this (mc)-contact grasp prototype, mc contacts must be generated on the new target object. In general, it would be desirable to match the prototype using a c-contact grasp with friction. This means that for each of c potential contacts on the target object, m samples must be taken to represent the range of contact wrenches that can be applied at that contact. In addition, the m samples of each contact of a new grasp must be aligned with the m samples from a contact of the grasp prototype. The choice of samples and the *phase* of this alignment are important, as they can drastically affect the cumulative quality measurement derived for the contact (Figure 7.7). In general, this process of approximating a range of wrenches at each contact is time-consuming and does not work very well.

The approach that was taken in the examples below is quite different. A frictionless grasp prototype is constructed in a space of reduced dimension. For the example prototype grasp shown above, the prototype is designed to cover four degrees of freedom in wrench

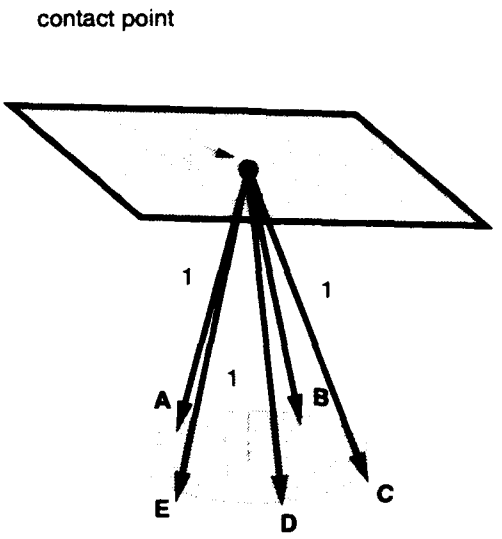


Figure 7.6: Sampled unit forces at a contact with friction. The convex hull formed from the contact wrenches corresponding to these sample contact forces approximates the boundary of the space of unit wrenches *possible* from this contact.

space without relying on friction. The grasp can easily resist forces perpendicular to the cylinder axis and torques that are not about the cylinder axis. A potential match to this prototype is evaluated by ensuring that there is sufficient friction for each contact wrench to point into this reduced dimensional space without slipping, projecting each contact wrench of the proposed grasp into this four-dimensional space in the way requiring the smallest coefficient of friction, and evaluating the resulting reduced dimensional grasp by matching it to the reduced dimensional prototype.

If the coefficient of friction is high enough to create a grasp that does not slip in this four-dimensional space, then the quality measure that is calculated for the grasp reflects the effectiveness with which the grasp will be able to respond to task wrenches in that four-dimensional space. The effectiveness with which the grasp can respond to any task wrenches in the other two dimensions of wrench space depends on the difference between the coefficient of friction available and the coefficient of friction required in order for the wrenches that make up the four-dimensional proposed grasp to lie within the friction cones at the contacts. If this difference is small, the hand will have to squeeze very hard to counter forces in these other two dimensions. Of course, this particular separation of dimensions into friction and frictionless dimensions is artificial and is only necessary for obtaining lower-bound grasp quality measures in the reduced dimensional space. If there is any doubt about the actual stability of a particular grasp, it is easy to test it more completely by sampling the friction cones at the contacts of a proposed grasp and evaluating the quality of that particular grasp in the full six-dimensional wrench space.

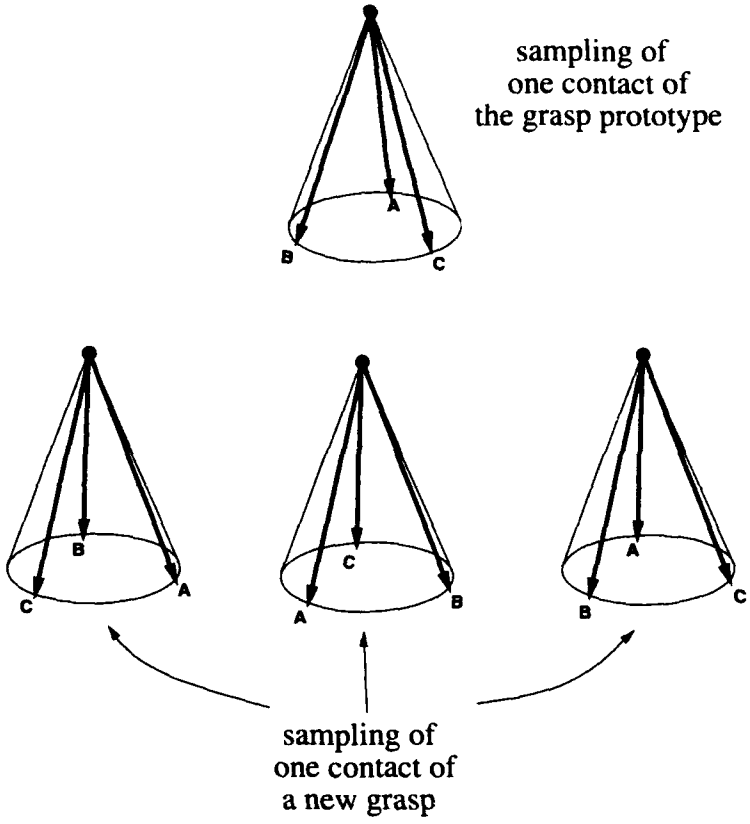


Figure 7.7: To estimate grasp quality from independent contact quality measurements, each sample contact wrench from a contact of a grasp prototype must be matched with a sample contact wrench from a contact of a proposed grasp. The ordering of this match affects the cumulative quality measure derived for the contact.

7.4 Grasps of Cylinders

This section shows that a cylindrical prototype grasp, such as that shown in Figure 7.1 can be used to solve two complex, related problems:

- Classification of possible grasps when obstacles are not present. The range of grasps matching the cylindrical prototype is illustrated for three target objects of increasing complexity.

- Avoidance of obstacles. Two of the three target objects are used to compare the range of grasps possible in a crowded workspace to the range of grasps possible when no obstacles are present.

To set up these problems, this section first analyzes the cylindrical prototype grasp in some detail. Then the two problems listed above are addressed. Finally, this section describes two grasps of real objects that were found by this grasp synthesis technique. These grasps were verified using the Salisbury hand.

7.4.1 A Prototype Cylinder Grasp

The prototype grasp used in the examples of this section is a grasp of a cylinder, similar to that shown in the two views of Figure 7.1. It consists of seven contacts: one contact on each of the two distal links of each of the three fingers and one contact on the palm.

The grasp is designed to span four dimensions of the complete six-dimensional wrench space without relying on friction. The four good dimensions are horizontal forces, and torques about the horizontal axes. Grasp quality measures and contact quality measures are computed using only these four dimensions.

The two dimensions of vertical force and torque about the vertical axis rely on friction. If there were no friction, the object would slide right out of the grasp. The examples of this section assume a coefficient of friction of 0.6. All contact forces must be able to generate contact wrenches that point into the four-dimensional space of horizontal forces and torque about the horizontal axes without exceeding this coefficient of friction at a contact. The projected contact forces and their corresponding torques are used to compute contact quality and grasp quality measures.

Independent Contact Regions of the Prototype

For the examples in this chapter, grasp quality is measured as defined in Expression 2.25. This grasp quality measure was designed to approximate resistance of a grasp to disturbance forces, and the wrench space of this task is given as a wrench space ball.

Using this task wrench space, the independent contact regions of the prototype object can be computed as for the two-dimensional prototype grasp of Chapter 4. The prototype grasp of that chapter is shown in Figure 4.1 and a corresponding set of independent contact regions is shown in Figure 4.5.

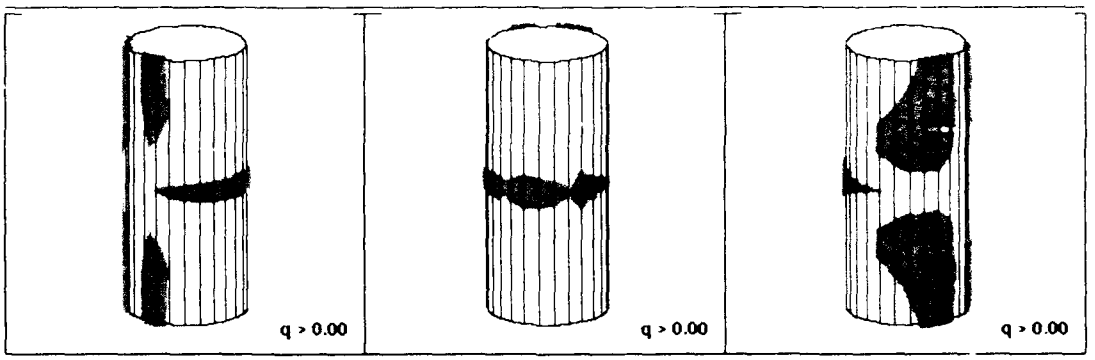


Figure 7.8: Positive quality contact regions corresponding to a 7-contact, 4-dimensional cylinder grasp. The figure shows three views of these regions when the cylinder axis is aligned with the prototype axis.

Results for the prototype cylinder grasp are shown in three views in Figure 7.8. This is a seven-contact grasp, and the seven regions corresponding to a contact quality measure greater than zero can be seen in the figure. The rightmost view shows two large regions. These correspond to the contacts on the distal link of the right and left fingers. The leftmost view shows the smaller regions corresponding to the middle link of the right and left fingers. The center view shows the three thin contact regions corresponding to the contacts on the two distal links of the thumb and the contact on the palm.

7.4.2 Aligning the Grasp Prototype to the Target Object

The cylindrical grasp prototype of this chapter has a problem of symmetry when aligning it to a new target object: it is symmetrical for rotation about the vertical axis of the cylinder. This means that while a new target object can be aligned to the cylindrical grasp prototype by placing the center of mass of the target object and the center of the grasp prototype at the same point and aligning the two cylinder major axes, there is still one orientation degree-of-freedom that must be resolved: relative orientation about these major axes.

Ideally, it would be sufficient to just sample this single orientation degree of freedom. Unfortunately, the thin strip of thumb and palm regions shown in Figure 7.8 proves to be a weak point with this grasp prototype, and it is necessary to allow some rotation of the prototype about an axis orthogonal to the axis of symmetry.

This rotation of the prototype axis away from a target object axis will affect the sizes and shapes of the independent contact regions on the target object. It is interesting to see what effect this has when these regions are computed for the prototype object. Figure 7.9 shows the results from tilting the prototype axis $\frac{\pi}{6}$ radians away from the cylinder axis in two opposite directions. This does not affect the sizes of the thumb and palm regions, but it does affect the sizes of the finger regions. One of the regions nearly disappears

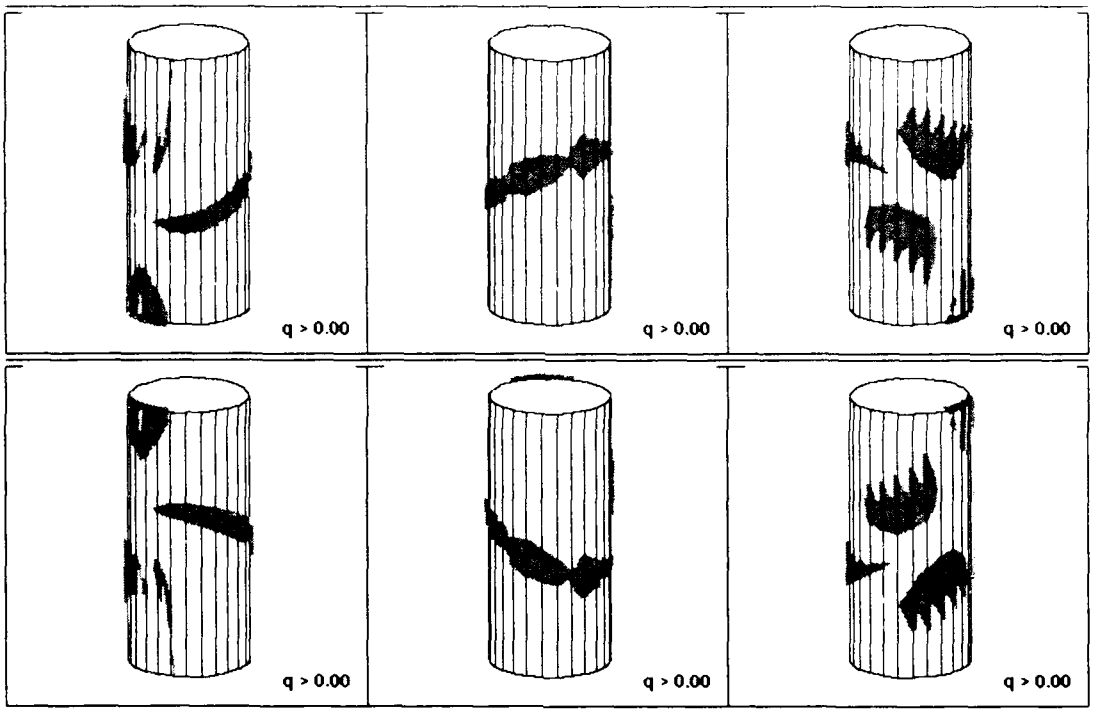


Figure 7.9: Positive quality contact regions corresponding to a 7-contact, 4-dimensional cylinder grasp. The rows show the regions that result when the cylinder axis is tilted $\frac{\pi}{6}$ radians away from the grasp prototype.

(the long, vertical, wispy region in the left hand figures). This is because the prototype axis passes very near that region, and sufficient torques cannot be generated to meet the contact constraints corresponding to that region. Two of the other regions (in the right hand figures) are smaller than in the neatly aligned prototype, and they have jagged edges. The slanted segments of these jagged edges run parallel to the prototype axis. Contact wrenches on one side of these segments point into the friction cone when projected into the four-dimensional "frictionless" space of the prototype, and contact wrenches on the other side of these segments point out of the friction cone after this projection. These figures illustrate some of the limitations of using this type of reduced-dimensional prototype in a situation that is less than ideal.

In the examples below, 180 alignments are sampled. These represent 90 right-handed grasps and 90 left-handed grasps at evenly spaced orientations about the cylinder major axis. Every third sample is tilted $\frac{\pi}{6}$ radians away from the cylinder axis. The sample following is tilted $\frac{\pi}{6}$ radians in the opposite direction. The sample following is not tilted away from the cylinder axis.

7.4.3 Extracting Solutions

The solution obtained from running the grasp synthesis algorithm is a set of optimal hand configurations distributed over the six-dimensional space of wrist configurations. In the examples below, only good wrist *positions* are illustrated. Figure 7.10 shows an example using the prototype object. Boxes represent horizontal slices through the three-dimensional Cartesian space processor grid, and the figure is read left to right, top to bottom, down the axis of the cylindrical target object. Grey regions represent the cylinder. White regions represent wrist positions from which some good grasp (a grasp with a quality greater than zero) can be achieved. Note that this is a three-dimensional projection of the complete six-dimensional solution space. White regions are wrist positions for which a good grasp can be achieved at *some* wrist orientation. To find out what orientation or orientations are represented and to extract the best solution found for a wrist position, it is necessary to query the processor representing that wrist position.

Each wrist configuration stores not only a boolean value indicating whether any good grasps are reachable from that wrist configuration, but also the lower-bound grasp quality measure of an optimal solution and an optimal hand configuration reachable from that wrist configuration. As might be expected from the coarse grid sampling and the large tolerances, the optimal hand configurations that are stored do not always fit the hand well. This is because each solution consists of a set of link configurations that can approximately, but not exactly, be fit together given the constraints of the hand kinematics. It is necessary to fit the hand model to this approximate solution, and this is done deterministically. If the fingertip positions and the wrist orientation are known, then a hand configuration can be determined. There is only one ambiguity in this solution with the kinematics of the Salisbury hand—does the distal joint bend up or down? This is decided using the expected position of this joint from the approximate solution. Unfortunately, this technique sometimes results in a hand configuration visibly different from the approximate solution used to find this hand configuration. This can be seen in the results that are presented below. There will be visible collisions between the hand and target object, and there will be cases where the hand does not seem to contact the object in exactly the right places. These problems could be eliminated by adding a local control step after the deterministic matching process. The fit of the hand to the intended contact points could be optimized and collisions could be avoided. With more processing power or more time, it would even be possible to do local optimization on the full grasp quality measure instead of its lower-bound approximation obtained from the independent contact quality measures.

Figure 7.11 shows an example solution format. This solution shows a graphical illustration of a grasp similar to the cylinder prototype in two views in the top row, and it shows the seven contact points of the grasp in the middle row. For this grasp, a grasp quality of 0.26 is guaranteed. The contact points are used to compute an actual grasp quality measure for the grasp. The actual quality found for this grasp is 0.41. The grasp is then used to form a new grasp prototype, and the regions corresponding to the new prototype and a grasp quality greater than 0.2 are shown in the bottom row. All solutions

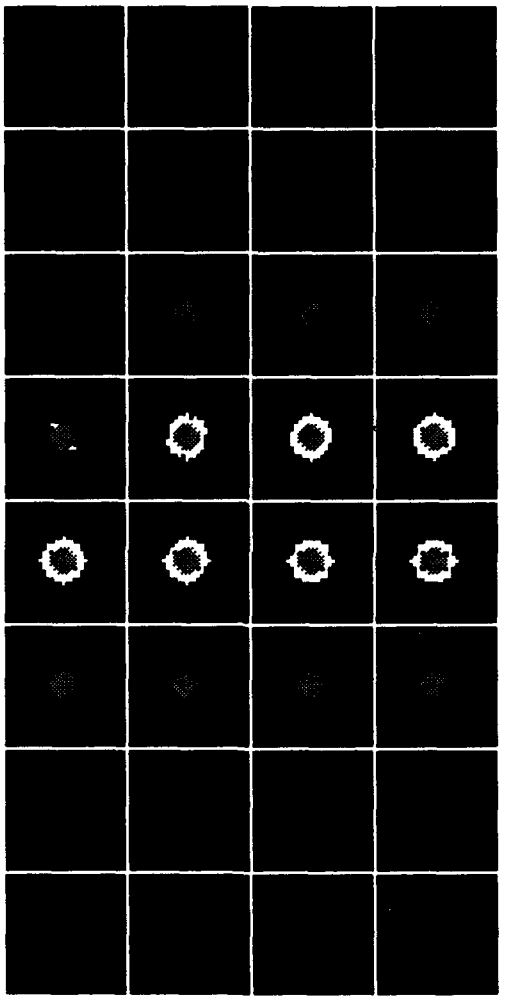


Figure 7.10: This figure shows wrist positions from which a good cylinder grasp can be achieved. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the cylinder axis. The grey object is the cylinder. The white areas represent good wrist positions.

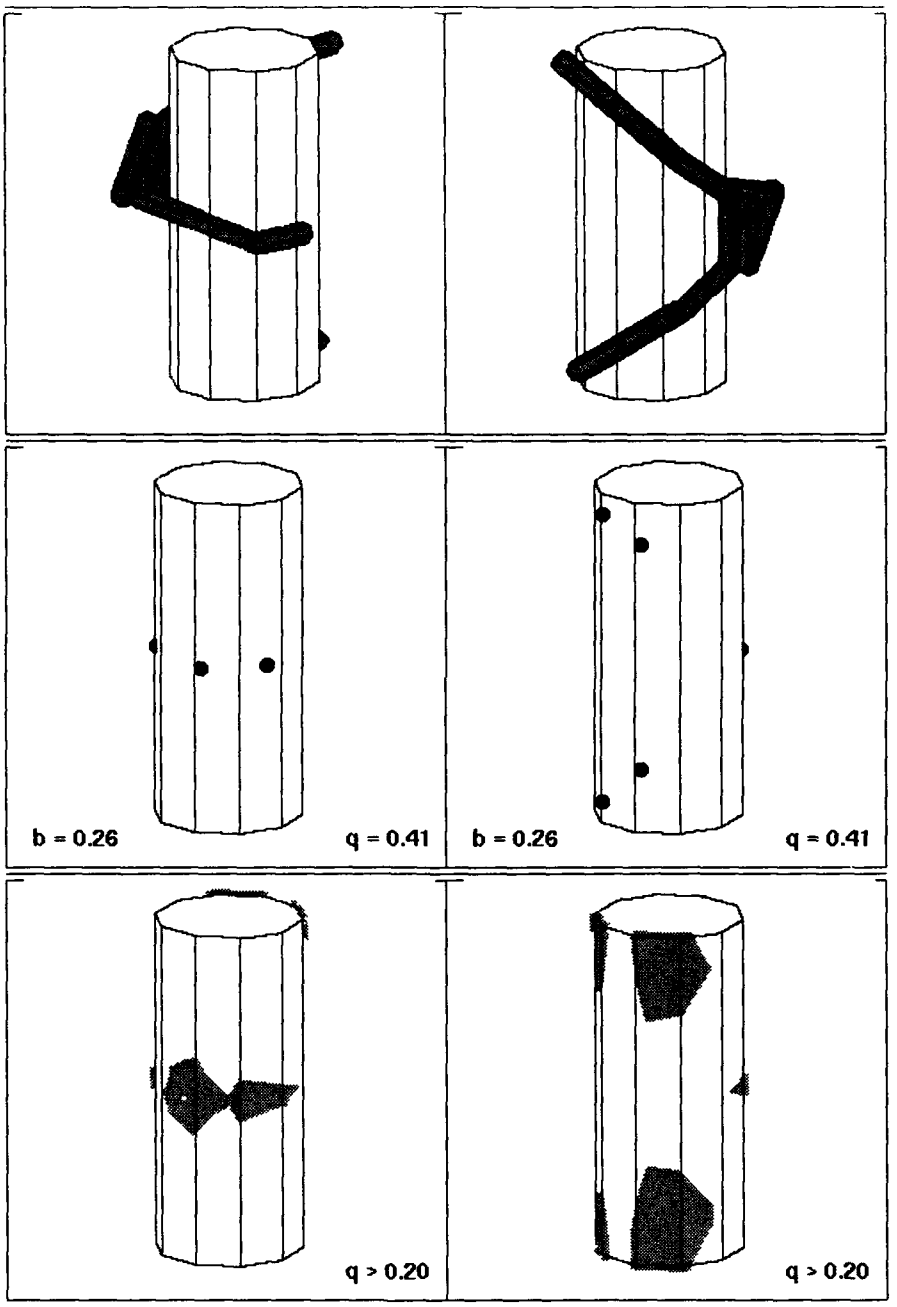


Figure 7.11: A match of the cylinder prototype to a cylindrical object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

illustrated will have this format.

7.4.4 Grasp Classification

This section presents some grasp classification results, using the cylinder grasp prototype that was described in detail above. Three target objects are tested.

Grasps of the Prototype Object

The first object is the prototype object—a simple cylinder. Figure 7.10 shows the wrist positions from which a grasp matching the prototype can be achieved. Many of the solutions in this space are very similar to the grasp prototype, such as that solution shown in Figure 7.11. In this solution, the prototype axis is exactly aligned with the cylinder axis. There is a continuous range of solutions of this type, covering about 1.5" vertically and the full range of orientations about the cylinder axis. There are also solutions corresponding to a prototype axis tilted with respect to the cylinder, but these solutions have a lower guaranteed grasp quality. One such solution is shown in Figure 7.12. It has a guaranteed grasp quality measure of 0.15 and an actual grasp quality measure of 0.25.

Grasps Requiring more Accurate Contact Placement

When the target object differs from the prototype object, the variety of good solutions may not be as large. Figure 7.13 summarizes the solutions available for a more complex object, a cylinder intersected by two toroidal objects. The solutions cover roughly the same vertical area, but they are much more sparse. This means that solutions will be more sensitive to contact placement in this example.

Figure 7.14 shows the best grasp found for this object. As with the cylinder, a range of solutions can be found about the target object axis. Figures 7.15 and 7.16 show two more solutions in this range. The three solutions are shown from the same points of view with respect to the target object. Notice that the placement of the thumb is very different in the three figures.

As with the cylinder, there are also solutions where the prototype axis is tilted with respect to the target object axis. Figure 7.17 shows one such solution. This grasp in general would not be presented as one of the best grasps, because there are many candidate grasps with higher guaranteed grasp quality measures in the solution space. Such a solution would only be extracted as a good candidate if these other solutions were somehow blocked.

Grasps Requiring more Accurate Wrist Placement

The solution in Figure 7.17 would not normally be extracted as a good grasp of the example object in that figure. This section presents an object for which this class of grasps is very important. A bar has been added through the center of the object to eliminate the best solutions of the previous example. To grasp this object, the hand must go over or under this bar. Figure 7.18 shows a summary of the results. The sampling is sufficiently sparse

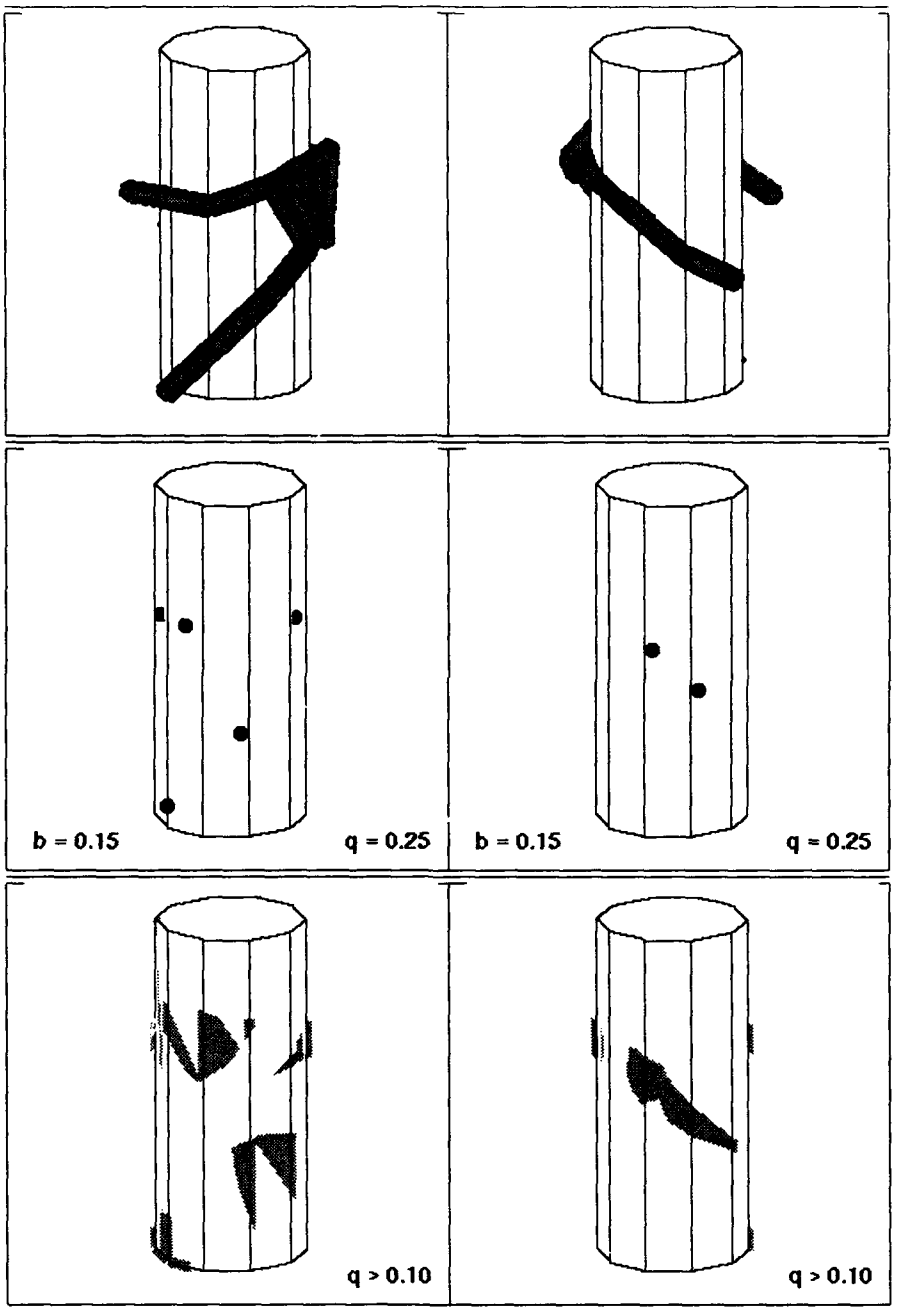


Figure 7.12: A match of the cylinder prototype to a cylindrical object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.1.

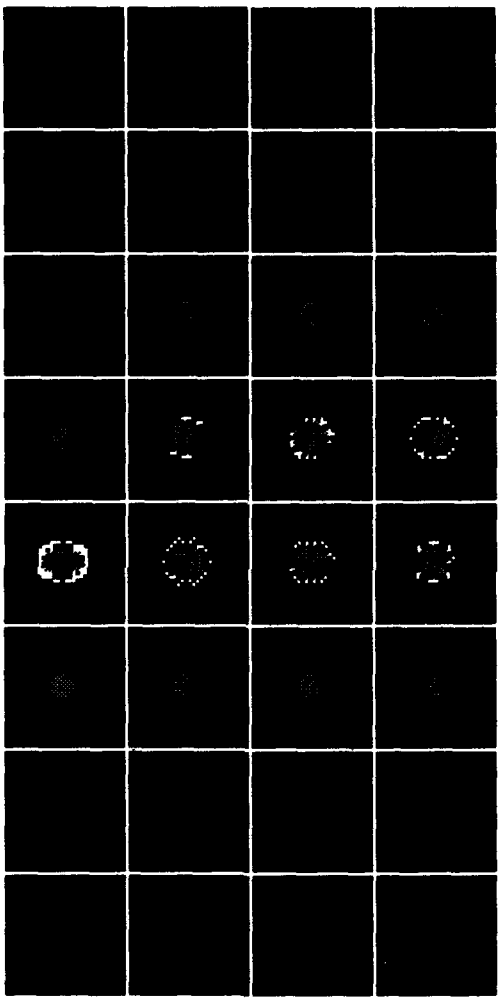


Figure 7.13: This figure shows wrist positions from which a good grasp of the more complex cylindrical target object can be achieved. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey object is the target object. The white areas represent good wrist positions.

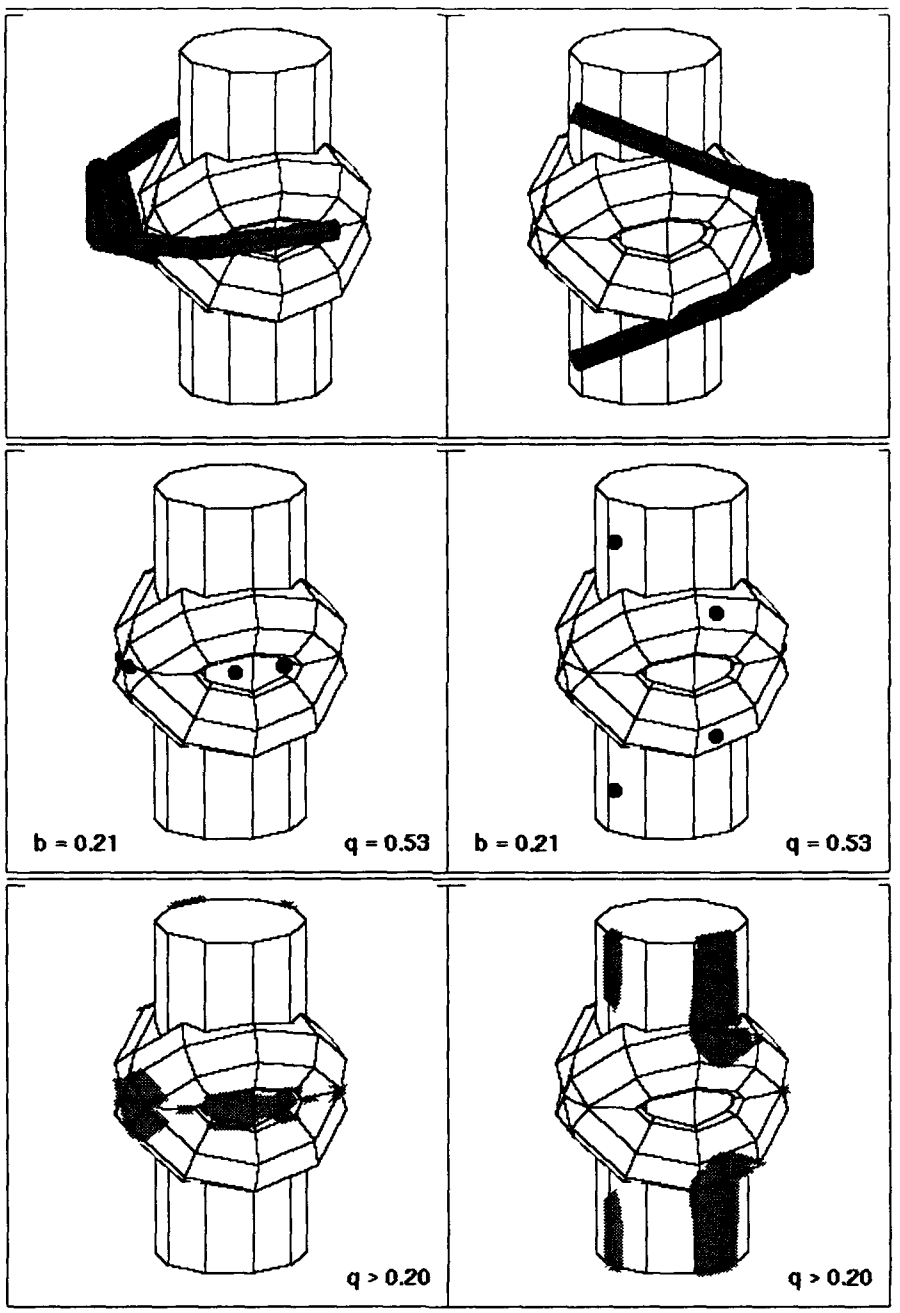


Figure 7.14: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

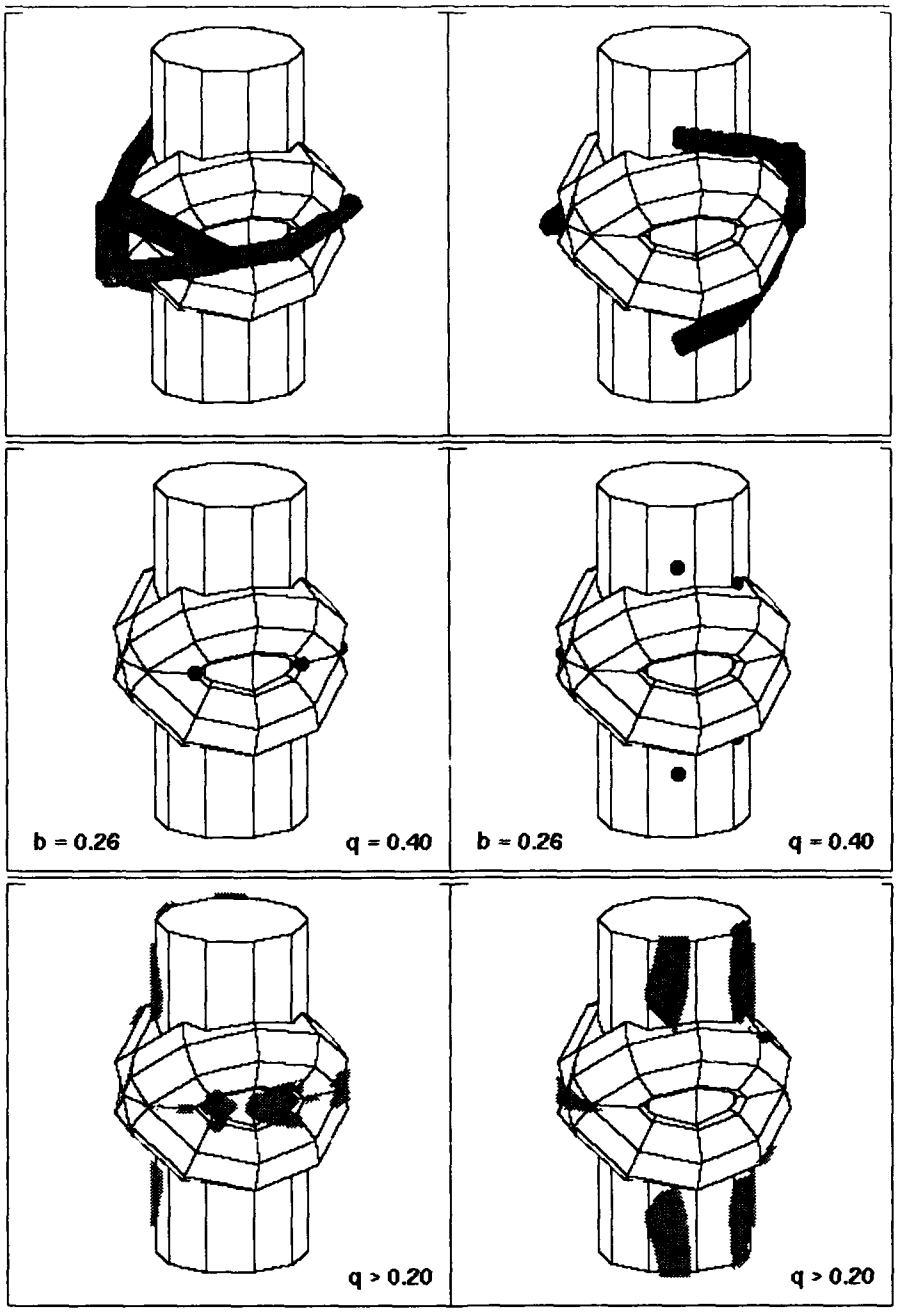


Figure 7.15: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

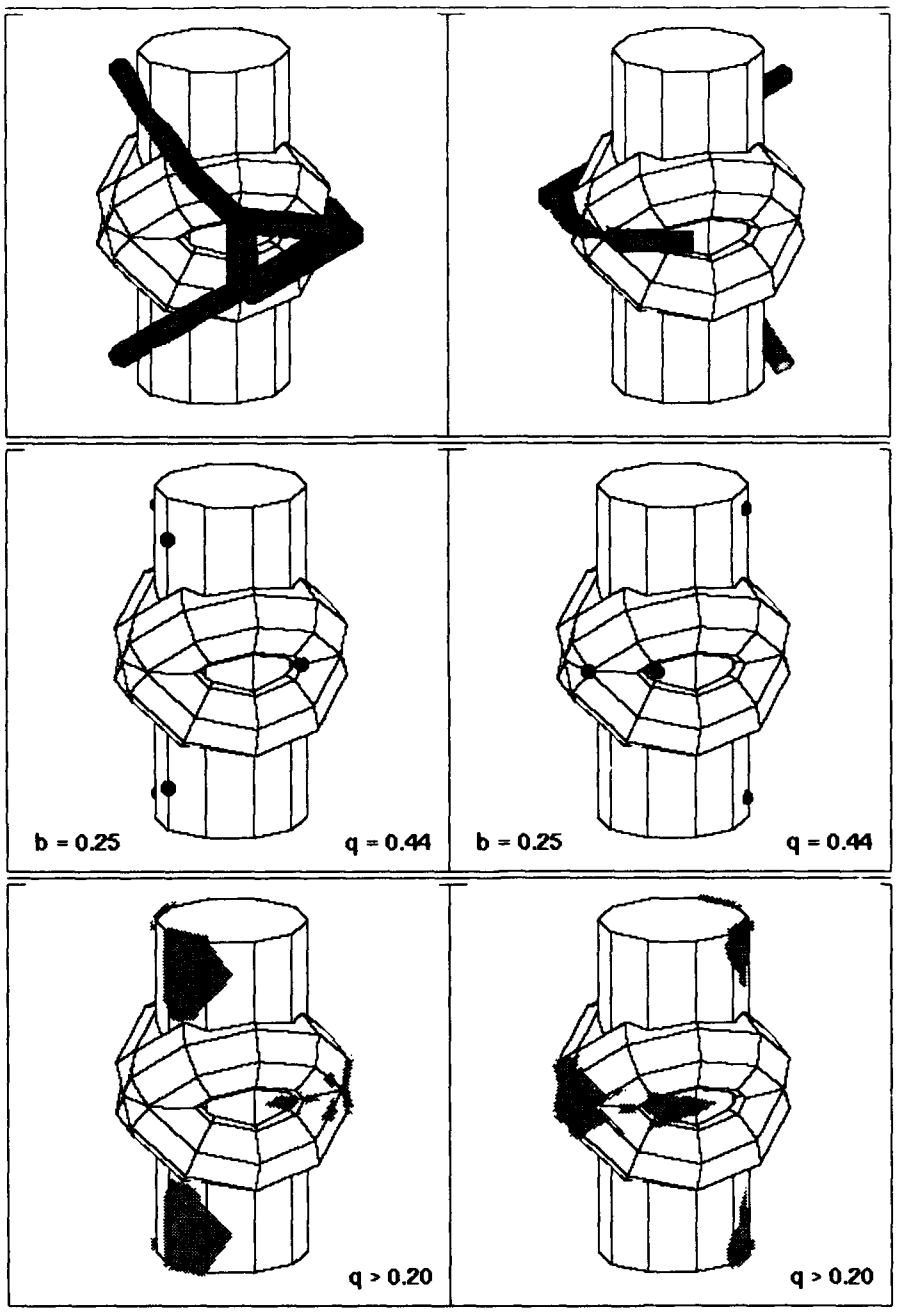


Figure 7.16: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

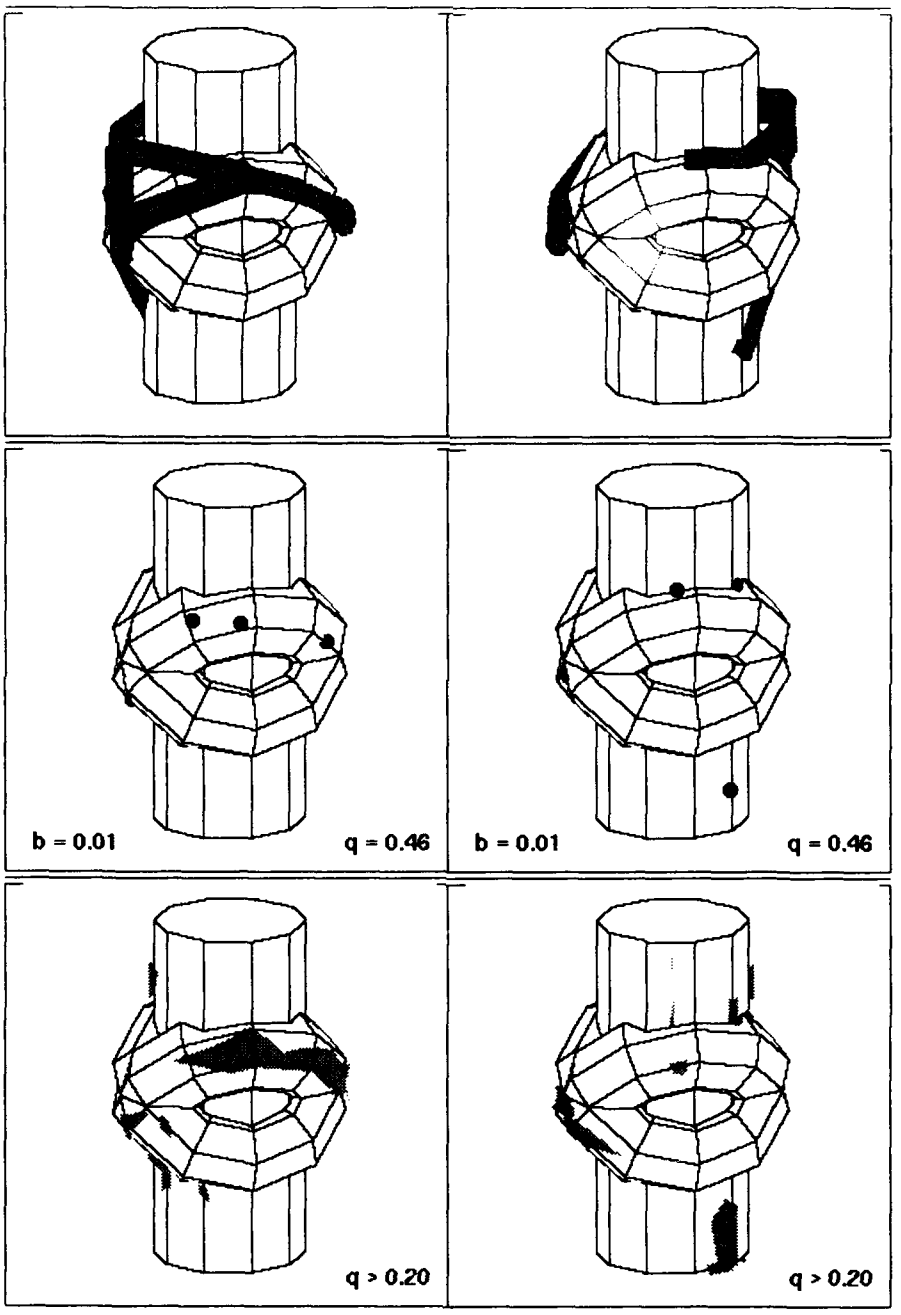


Figure 7.17: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

and the grasp sufficiently sensitive that only a few solutions are identified. The solution space can be understood fairly well by examining these solutions, however.

Figure 7.19 shows the best solution found. Although the sampling algorithm does not find all qualitatively different variations of this solution, it is clear that there will be eight: the object has two symmetrical halves; the hand can go over or under either bar; and it can do this left or right handed.

For each of these variations, there is a small range of solutions. Figures 7.20 and 7.21 illustrate solutions in other parts of this range. All three grasps are shown from the same views. Notice the variation in placement of the thumb with respect to the object cavity.

7.4.5 Obstacle Avoidance

The next few examples show how the solution technique presented in this chapter can be used to find grasps that avoid obstacles in the environment. An environment was constructed that had eight randomly placed, roughly spherical obstacles. Figure 7.22 shows a summary of the results in this environment for the cylindrical target object. The new grey areas are areas occupied by the obstacles. Compare this to Figure 7.10. The symmetrical pattern of Figure 7.10 has been broken by the introduction of obstacles into the environment.

Figures 7.23 and 7.24 show two of the solutions in this range. It is difficult to visualize, but both of these solutions are free of collisions. This example points out the insufficiency of using this algorithm alone, however. Although the grasps are collision-free, it is difficult to imagine that the hand could successfully extract the object from this environment without moving at least some of the obstacles.

Figure 7.25 shows results for the more complex cylindrical object in the same environment. Compare this figure to Figure 7.13. The obstacles have eliminated all but a small range of solutions.

Figure 7.26 shows one solution within this range. This is very similar to the solution shown in Figure 7.23.

7.4.6 Experiments

Two real objects—a toy plane and a bubble gun—were modelled (Figure 7.27) so that the resulting grasps could be verified using the Salisbury hand. Both objects were treated as cylinders, with the major axis of the plane passing from the front of the plane to the back of the plane through the body, and the major axis of the bubble gun passing through the centers of the three spheres making up the barrel of the gun. The center of mass chosen for the bubble gun reflected only the masses of these three spheres. The cylindrical parts of these two objects have much smaller diameters than the example cylinder. The plane diameter has a maximum value of 2.5 inches, and the bubble gun diameter a maximum of 2.25 inches, while the diameter of the example cylinders was 3.5 inches. Nevertheless, the same grasp prototype was used for these new objects. Figures 7.28 and 7.29 show the results, which are sparse for both objects.

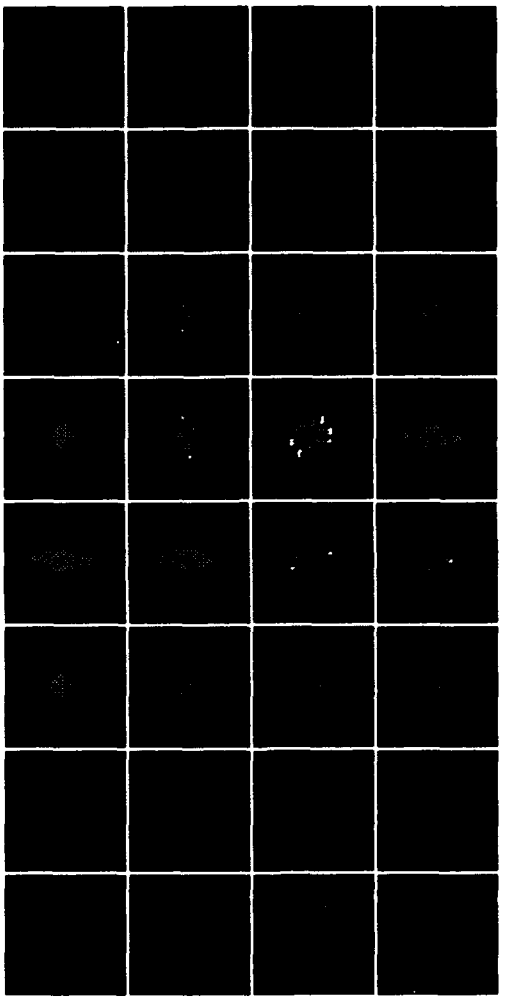


Figure 7.18: This figure shows wrist positions from which a good grasp of an even more complex cylindrical target object can be achieved. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey object is the target object. The white areas represent good wrist positions.

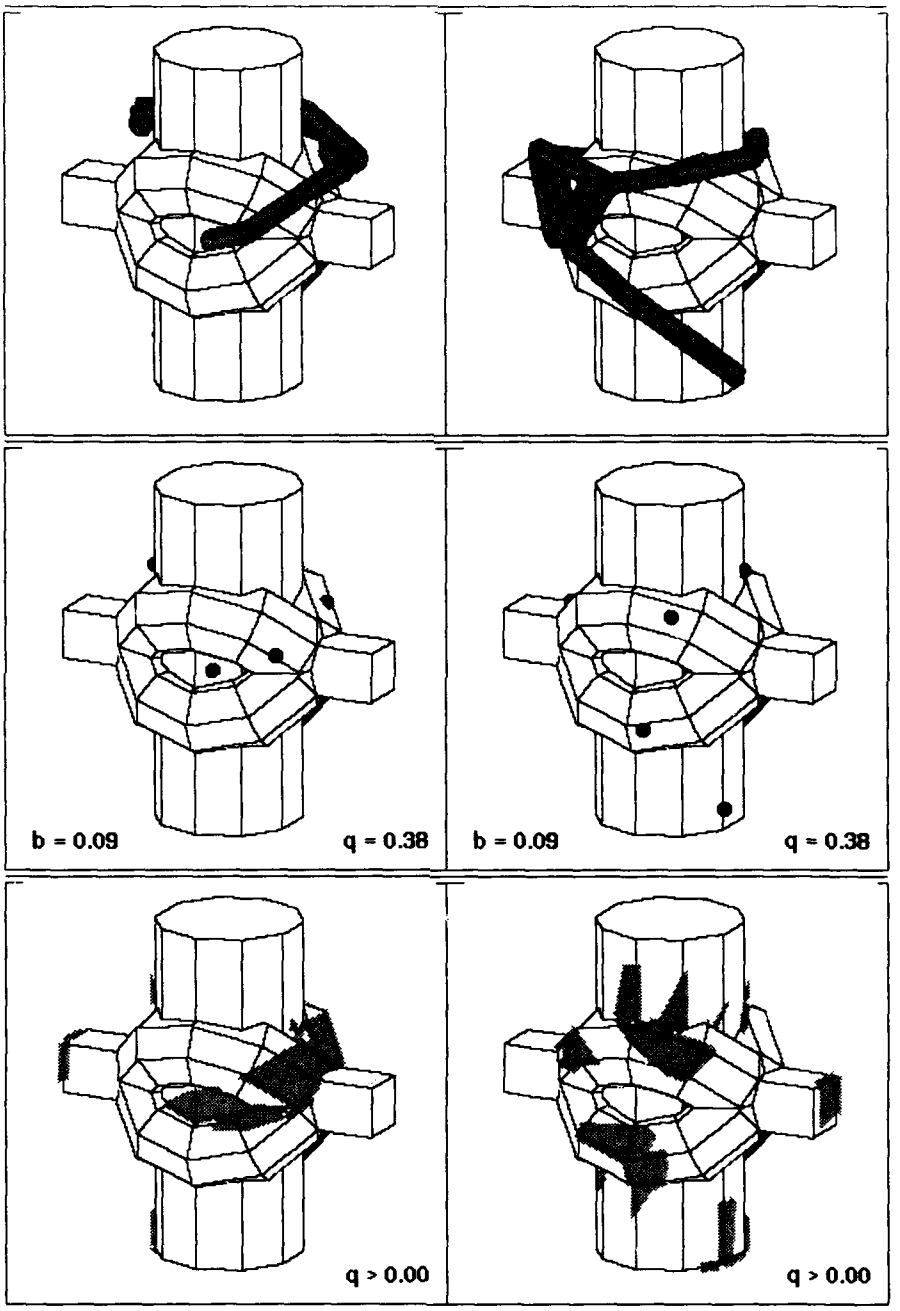


Figure 7.19: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.2.

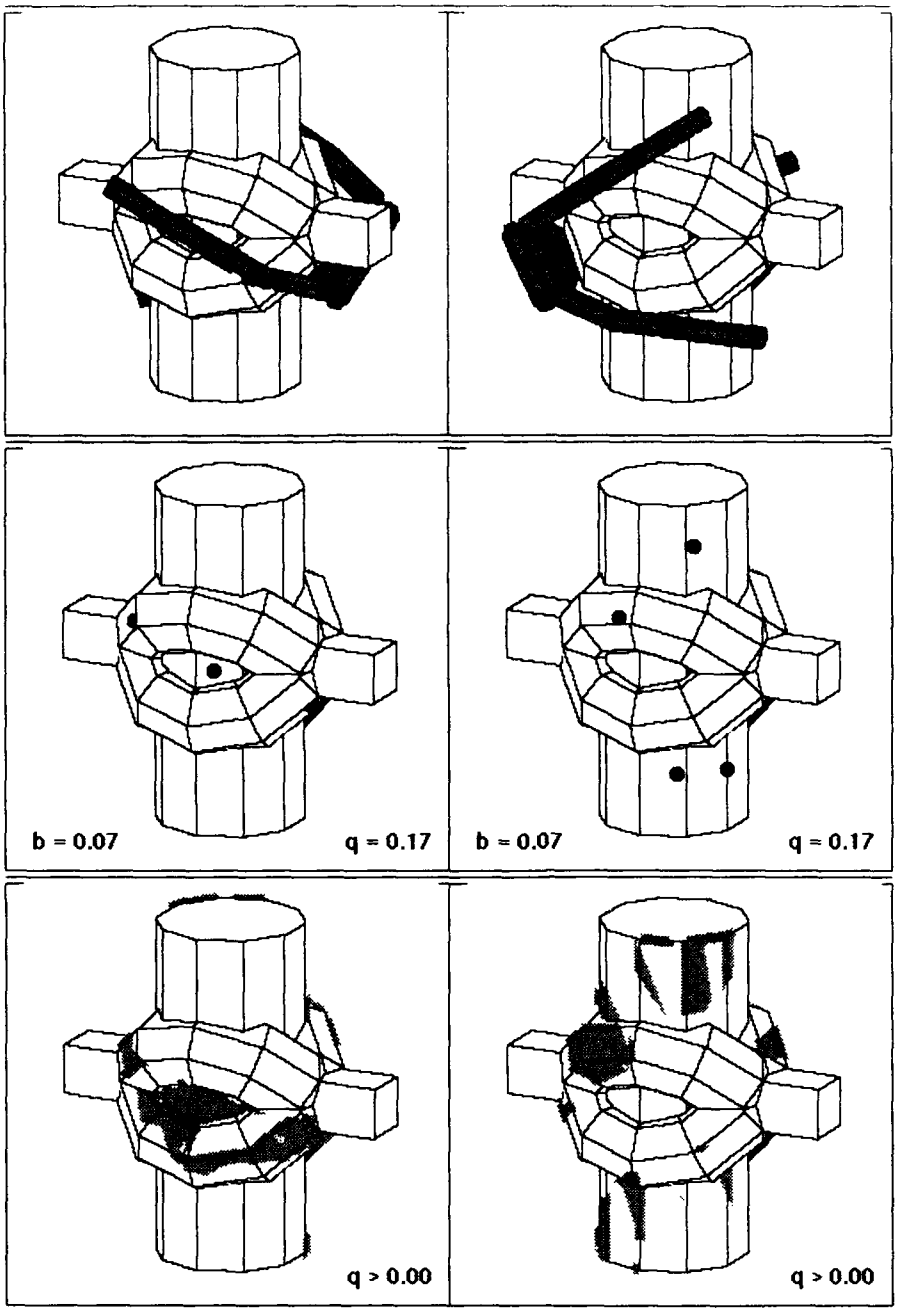


Figure 7.20: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.

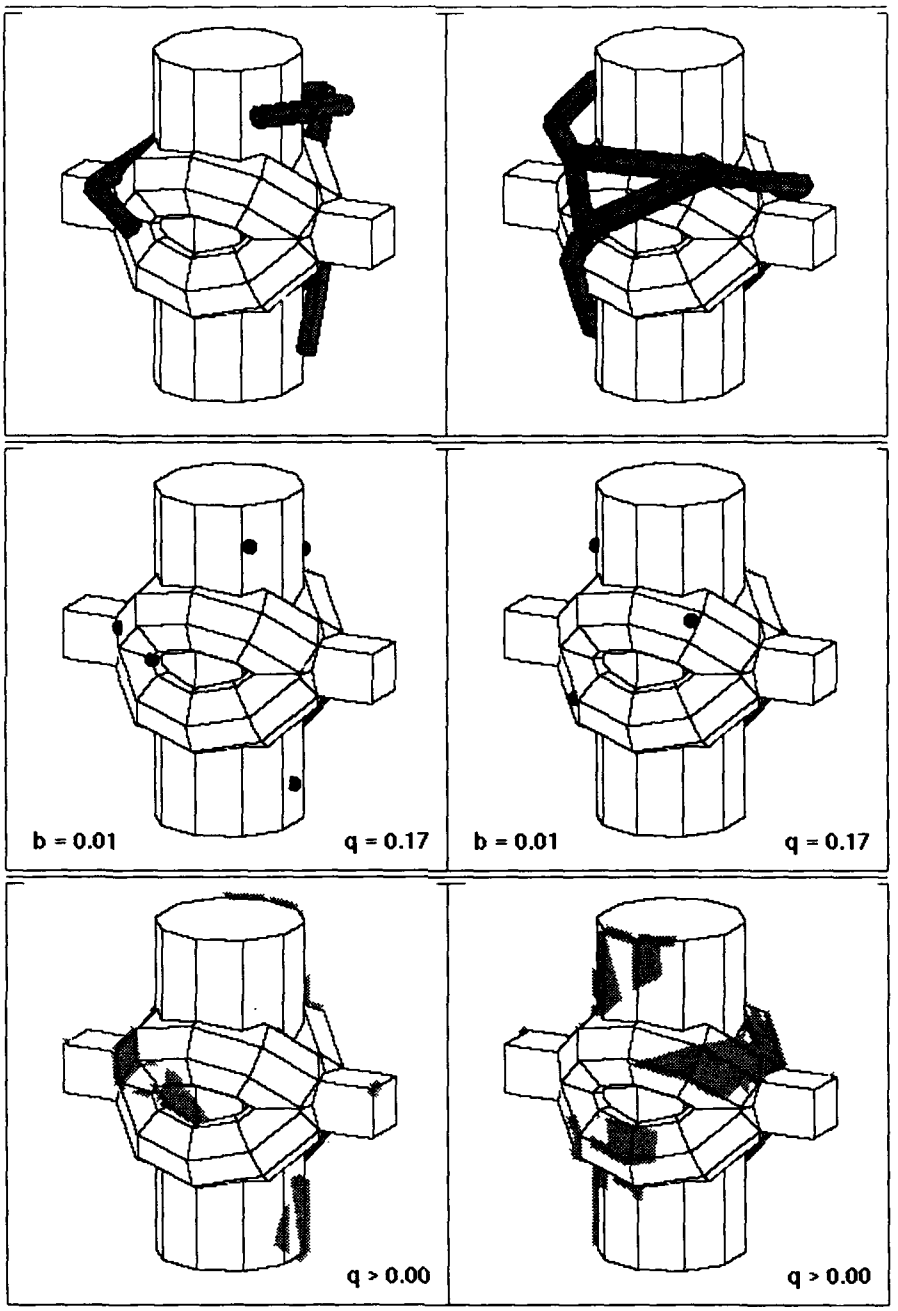


Figure 7.21: A match of the cylinder prototype to a more complex object. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.

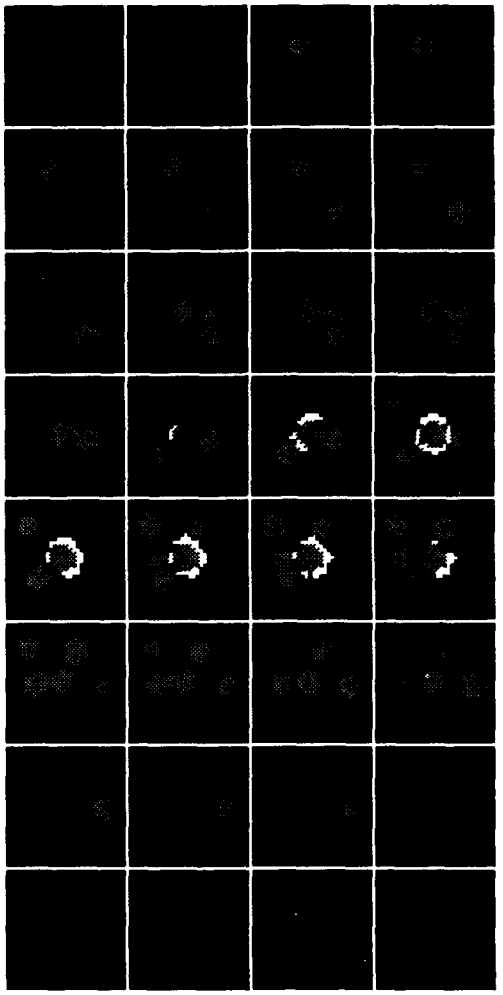


Figure 7.22: This figure shows wrist positions from which a good grasp of the original cylinder object can be achieved in an environment with obstacles. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey objects are the target object and the obstacles. The white areas represent good wrist positions.

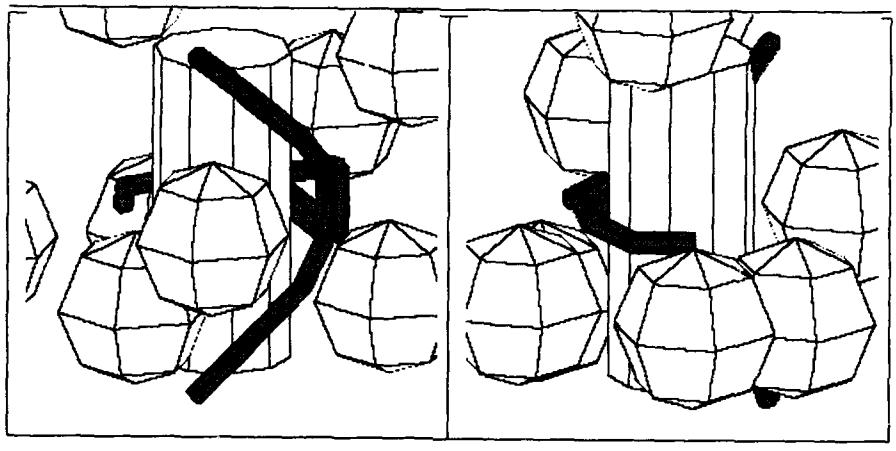


Figure 7.23: One collision-free grasp of the cylindrical target object found in a crowded environment.

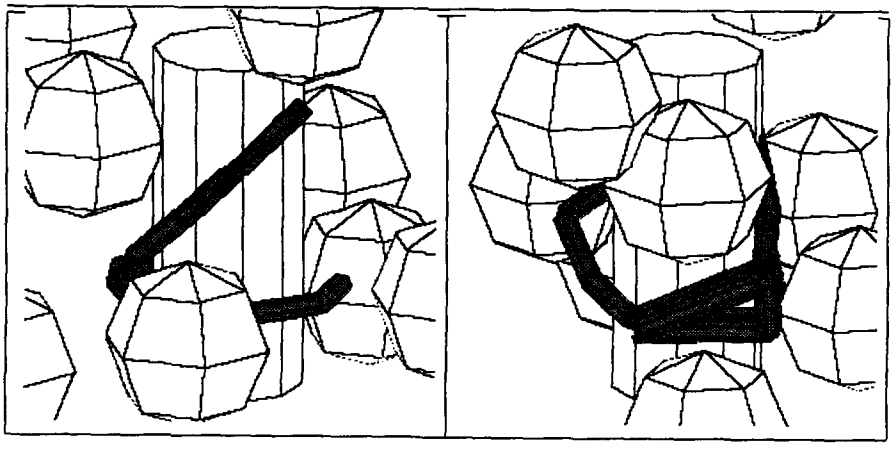


Figure 7.24: A second collision-free grasp of the cylindrical target object found in a crowded environment.

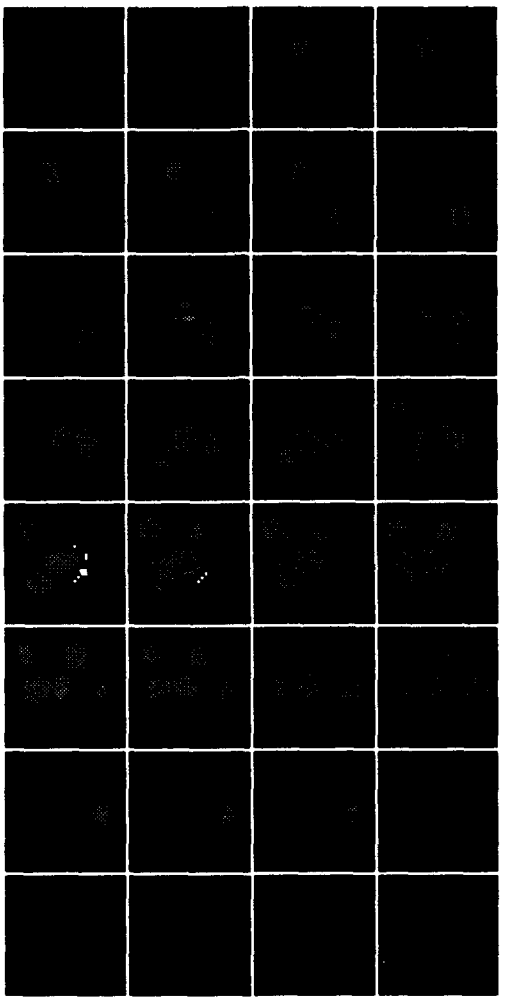


Figure 7.25: This figure shows wrist positions from which a good grasp of a more complex target object can be achieved in an environment with obstacles. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey objects are the target object and the obstacles. The white areas represent good wrist positions.

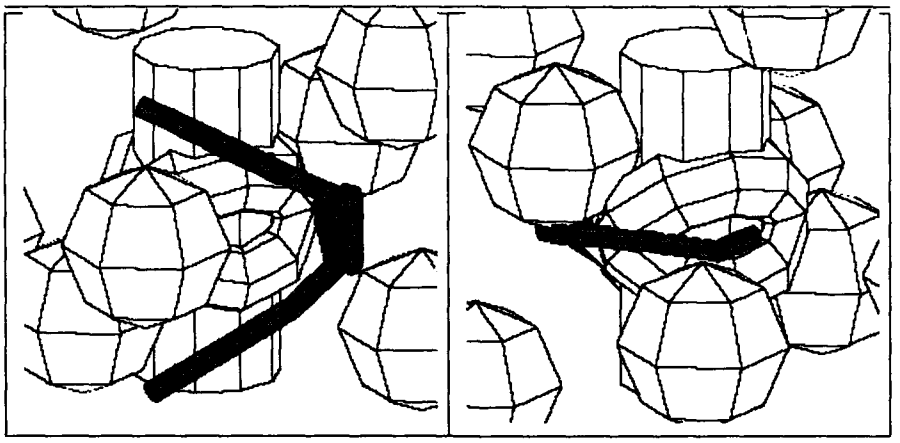


Figure 7.26: One collision-free grasp of the more complex target object found in a crowded environment.

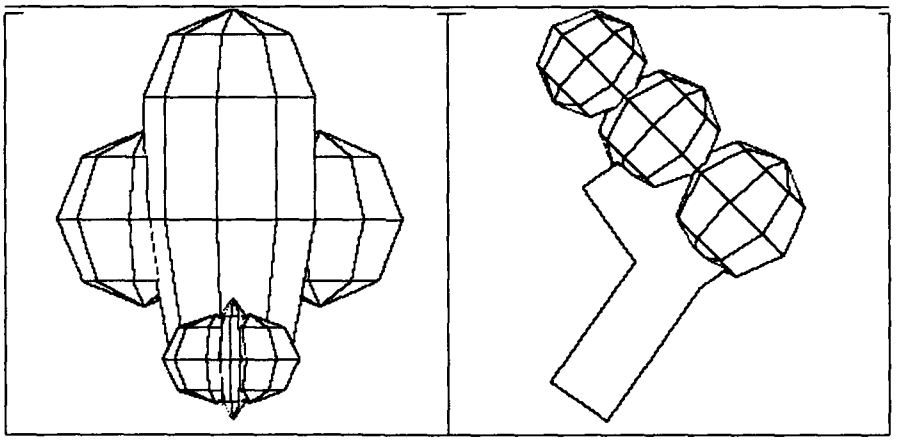


Figure 7.27: Models of two real objects—a toy plane and a bubble gun.

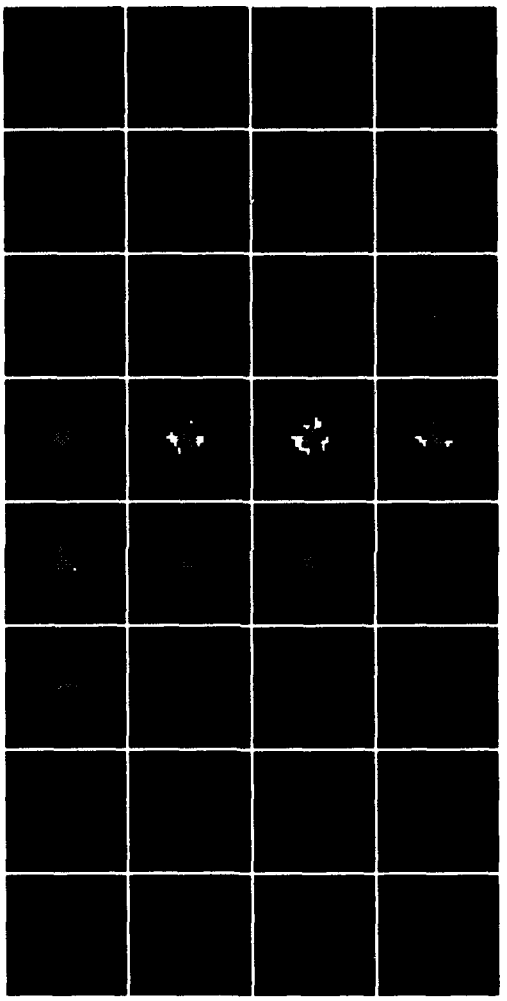


Figure 7.28: This figure shows wrist positions from which a good grasp of the toy plane can be achieved. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey object is the target object. The white areas represent good wrist positions.

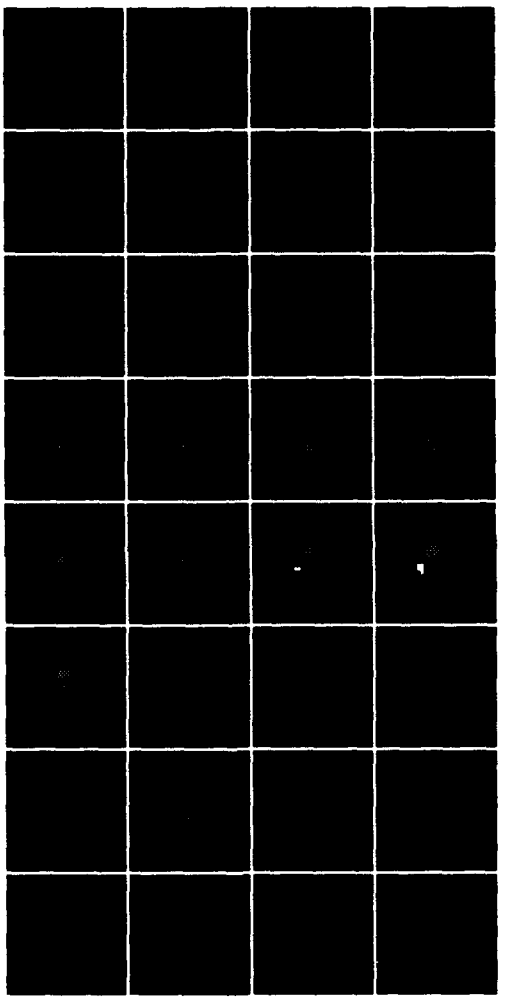


Figure 7.29: This figure shows wrist positions from which a good grasp of the toy gun can be achieved. Each box represents a horizontal slice of three-dimensional Cartesian space. The slices are read left to right, top to bottom, down the target object axis. The grey object is the target object. The white areas represent good wrist positions.

In the grasp of the plane, the fingers must get around the two wings. There are eight possible grasps, each with a small amount of room to move, much as in the example shown in Figure 7.19. Figure 7.30 shows one of these solutions, a left-handed grasp, and Figure 7.31 shows two views of the hand grasping this object in the right-handed permutation. This was found to be an easily achievable and very solid grasp.

In the grasp of the bubble gun, there is only a small range of solutions because the robot is forced to use the trigger of the gun to expand the effective radius of the barrel of the gun by a small amount. A variable radius prototype cylinder grasp would have helped here. Figure 7.32 shows one solution that was found. Figure 7.33 shows the hand grasping this object. This was also a solid, easily achievable grasp.

7.5 Summary

This chapter has shown a variety of cylinder grasps, all found by applying the same cylindrical grasp prototype to a set of new target objects. This set of examples demonstrated that this grasp prototype is flexible with respect to variation in target object geometry. The cylinder with a bar through it (Figure 7.19) and the airplane (Figure 7.30) both required grasps different in appearance from the original prototype grasp.

In addition, this set of examples showed how the global overview of the solution space provided by the grasp synthesis algorithm, as shown in Figure 7.22, for example, can be very helpful in determining the effect of variation in the target object geometry on the shape of the solution space (compare Figures 7.13 and 7.10) and the effect of variation in the geometry of the environment on the shape of the solution space (compare Figures 7.22 and 7.10). This information provides a good starting point for a more complete algorithm that must find an arm configuration that is free of collisions and compute a collision-free path into a good solution. It is only worthwhile to look for complete solutions where the *hand* is free of collisions and can achieve a good grasp of the target object.

Now that these examples have been presented, the goals described in the beginning of Chapter 5 can be reviewed:

- **Find optimal grasps.** A hand configuration having a globally optimal lower-bound grasp quality measure (as described by Expression 3.18) can be easily extracted from the solution space. This is not in general the globally optimal grasp as defined by the grasp quality measure of Expression 2.25, for example, but it is guaranteed to be a high-quality grasp.

- **Evaluate how easily a grasp can be achieved.** Once a grasp has been selected, this grasp can be used to find good regions of contact placement for each contact of a grasp (e.g. see the bottom row of Figure 7.11). The sizes of these regions provide a measure of how easily a matching set of contacts can be achieved.

- **Measure the effect of obstacles in the environment.** By comparing the size and shape of the space of good wrist configurations for a grasp of a given target object

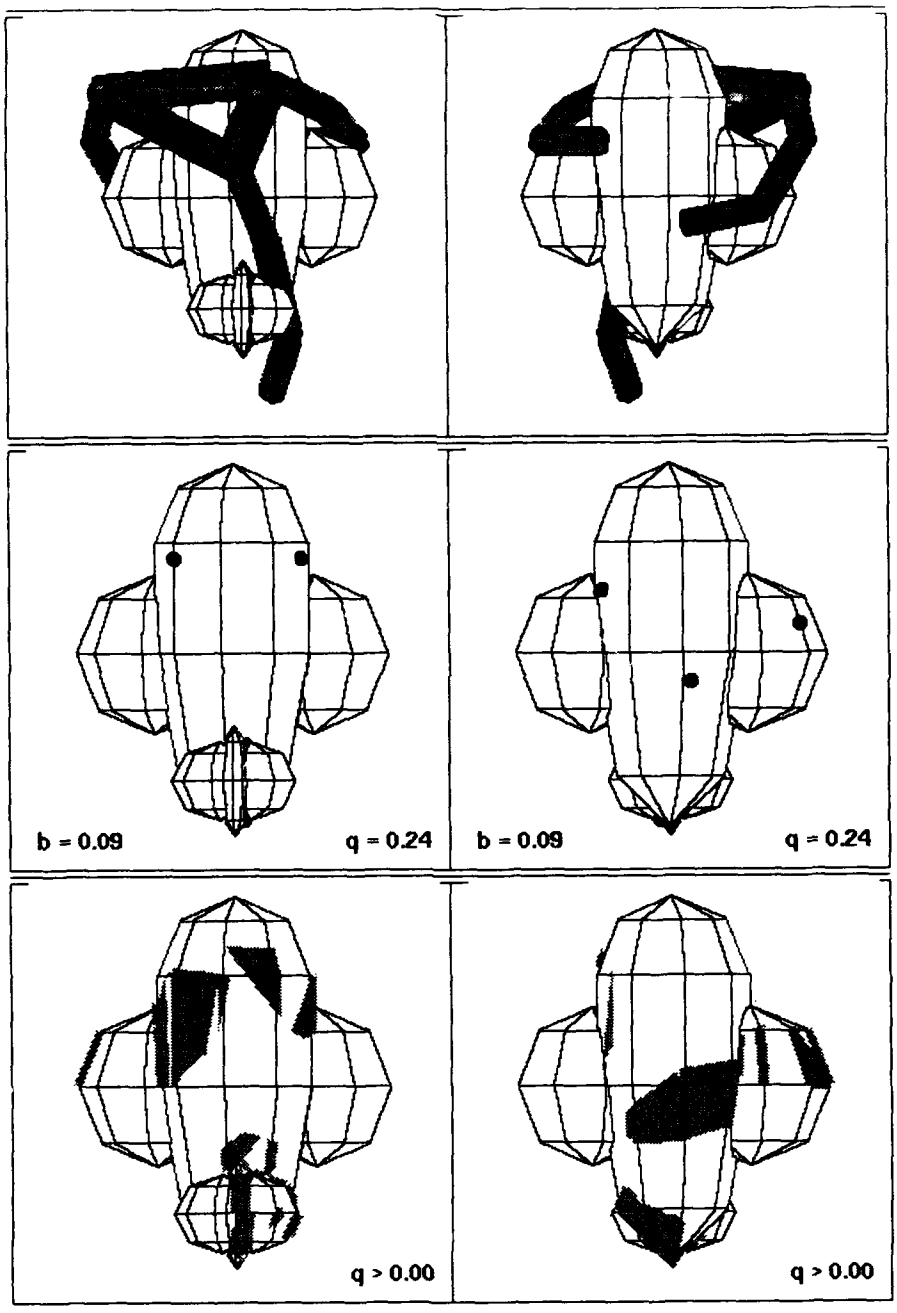


Figure 7.30: A match of the cylinder prototype to the toy plane. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.

Figure 7.31: Two views of the Salisbury hand grasping the toy plane.

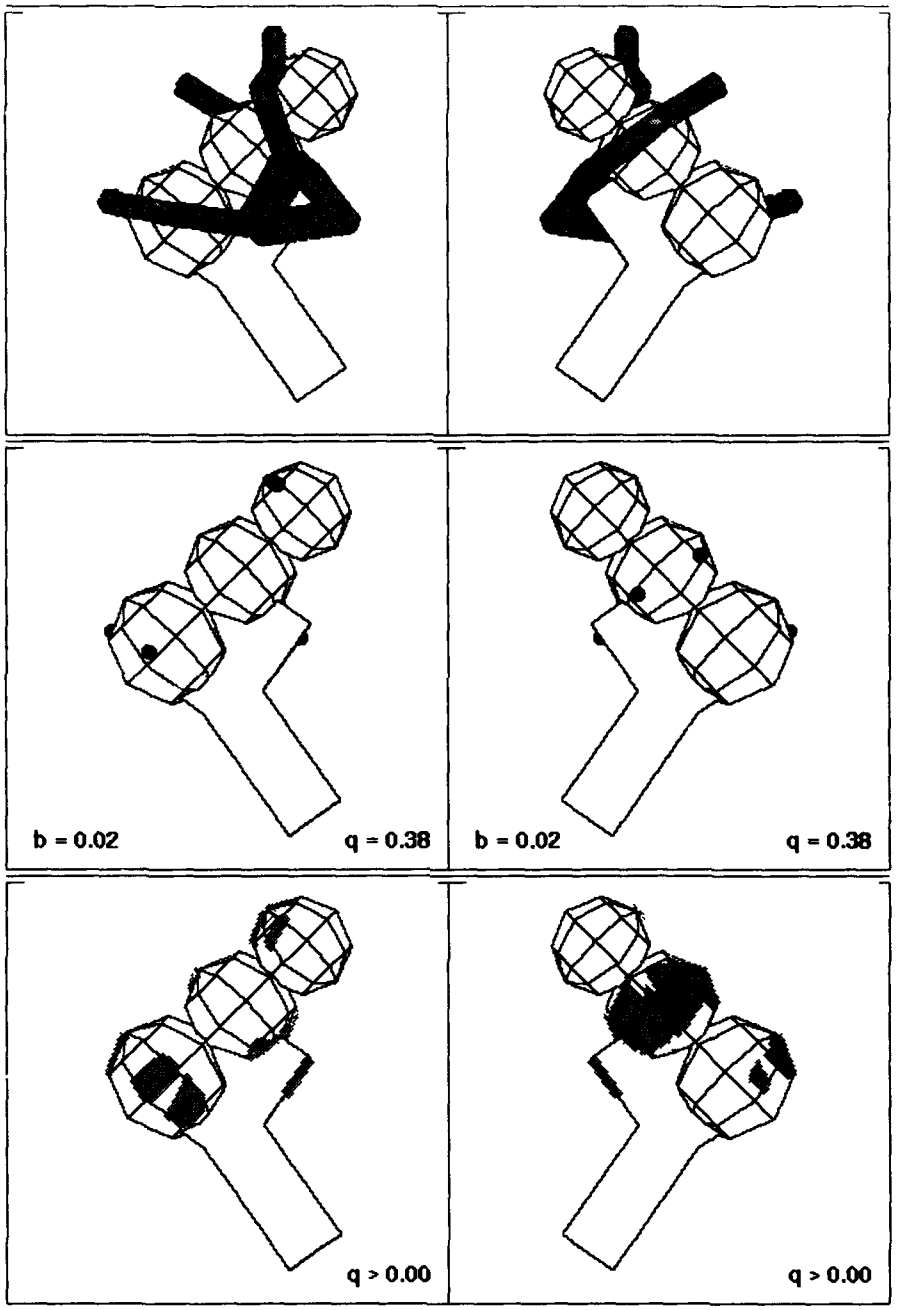


Figure 7.32: A match of the cylinder prototype to the toy bubble gun. The top row shows the grasp, the middle row shows the contact points, and the bottom row shows the contact regions calculated for this grasp and a quality measure of 0.

Figure 7.33: Two views of the Salisbury hand grasping the toy bubble gun.

in environments with and without obstacles (e.g. compare Figure 7.10 to Figure 7.22) the effect of obstacles on this projective solution space can be determined.

- **Measure the effect of the target object geometry.** By comparing the size and shape of the space of good wrist configurations for grasps of two different target objects in an environment without obstacles (e.g. compare Figure 7.13 to Figure 7.10) the effect of variation in the target object geometry can be determined.

- **Select good values for parameters of symmetry in a grasp.** By sampling the solution space over the parameters of symmetry of a grasp (such as symmetry in the orientation of the cylindrical grasp prototype about a target object major axis), good values for these parameters of symmetry can be selected. The difference between Figures 7.22 and 7.10, for example, highlights directions from which the hand might approach the cylinder object in the environment with obstacles.

In summary, although some compromises have been made—only a projective solution space is computed, and the grasp quality measures optimized are lower-bound estimates of the real thing—the grasp synthesis technique presented here does address all of the capabilities described as goals in the beginning of Chapter 5. This technique can be used to extract good solutions as well as to obtain a global overview of the solution space and measure the effect of geometric variations on this solution space.

Chapter 8

Summary

This report has addressed the problem of applying standard grasps in non-standard situations, situations where the geometry of the target object or the geometry of the environment differ from the geometry represented by the standard grasp. A grasp synthesis algorithm was presented that takes as input a single example grasp and produces as output a set of hand configurations representing good grasps of a new target object. These grasps are guaranteed to be free of collisions and suitable for the intended task.

The technique that was used to generate the solution space consists of three steps:

1. generalizing the example grasp for a given task,
2. applying this generalized grasp to a new target object to obtain good sets of contacts on that target object, and
3. finding collision-free hand configurations from which these good contact sets can be reached.

These three steps are reviewed here.

Generalizing the Example Grasp

The first step of the process is to extract the contacts of the example grasp and generalize this contact set so that it reflects a space of contact sets guaranteed to be appropriate for the intended task. The generalization process, which was described in Chapter 3, provides the flexibility necessary to accommodate a variety of target object and environment geometries. The generalization step is performed using the wrench space description of the grasp prototype, and not a position space description, or the *shape* of this prototype. This means that a generalized grasp description includes only constraints relevant for the given task, which is also expressed as a space of wrenches.

Two features of this grasp generalization process are especially interesting. First, it is designed to work equally well for any given task, as long as the task can be expressed as a space of wrenches applied to the target object. The example grasp can be generalized for any given task, no matter how complicated, in an offline process. This offline process

involves constructing a wrench space convex hull from the example grasp and determining the scale at which the entire task wrench space fits within this convex hull. The time required to execute this offline process is proportional to the complexity of the task, but the result is not. The process of matching a proposed new grasp to the generalized grasp can be accomplished by minimizing a small number of dot products for each contact of the grasp. The complexity of the matching process does not vary with the complexity of the task.

A second interesting feature of the grasp generalization process is that it allows contacts to be optimized for measures of contact quality and robustness to contact placement without knowing the values of the other contact wrenches of a grasp. This is possible because these other contact wrench values are approximately defined by the generalized grasp description. The ability to locally optimize the placement of each contact of a grasp allows the search for good contact sets to be parallelized over these contacts.

Finding Good Contact Sets

The second step of the process is to apply the generalized grasp description to the new target object for various alignments of the grasp prototype to that target object. Good sets of contacts on the new target object are extracted from this process. Once an alignment of the grasp prototype to the target object has been specified, identification of good contact sets is very simple, requiring only the computation of a few dot products at sampled contacts with the target object features. Any geometric information that is available can be used to simplify the task of selecting good alignments to test.

Finding Good Hand Configurations

The third step of the process is to search the space of hand configurations using a parallel, dynamic programming algorithm. The result is a representation of the space of wrist configurations from which good sets of contacts can be reached. This process produces a global overview of the space of good solutions. This global overview can be used to determine the effect of variation in the target object geometry on the size and shape of the solution space. It can also be used to determine the effect of obstacles in the environment on the size and shape of the solution space.

In addition to providing a global overview of the space of good solutions, this parallel optimization algorithm highlights a set of optimal hand configurations associated with good regions of wrist configuration space. These hand configurations can be examined to obtain a more complete idea of the types of solutions available than can be obtained by examining the size and shape of the space of good wrist configurations. This topic was explored in the many examples of Chapter 7. A hand configuration can be selected from this set to be used as the grasp to be achieved by the robot hand.

In addition, if intermediate results are retained from executing the parallel optimization algorithm, a search can be performed through the complete space of hand configurations. This search can be guided by the lower-bound grasp quality measures used in the

optimization, so unpromising areas of this very large space can be avoided. This search requires no additional calculation, only repeated references of local processor memory.

The grasp synthesis problem is extremely complex when the constraints of target object geometry, environment geometry, hand kinematics and geometry, and a task description are considered. Even so, the use of a standard grasp was shown in this report to make the grasp synthesis problem almost tractable. A global overview of the space of good solutions could be generated for grasps using a cylindrical grasp prototype in approximately one hour on available hardware. The global nature of the solution was very satisfying because it provided a good summary of the effect of variations in target object geometry or environment geometry on the size and shape of the solution space. The use of the simple cylindrical grasp prototype as described in this report was shown to be sufficiently flexible to apply to a wide range of target object and environment geometries.

8.1 Future Work

In order to round out this work, a number of additional topics must be considered, ranging from better consideration of friction and complex contacts to consideration of the preprocessing and postprocessing steps needed to form a complete grasping system. These topics are considered in the sections below.

8.1.1 Curved Objects

No curved objects were used as prototype objects, and no grasps of curved objects were synthesized in this report. Nevertheless, curved objects do not pose any new problems for this grasp synthesis technique. A point contact on a curved surface has a unique local normal. This local normal can be used to compute the frictionless forces and torques that can be applied at the contact and to compute the contact quality measure for the contact, just as with point contact on a planar surface. Because the algorithm proposed in this report uses a parallel distributed description of local target object geometry (Chapter 6), the only additional work required to accommodate curved objects is the work required to load the parallel computer with this geometric information.

8.1.2 Friction and Complex Contacts

Chapter 7 dealt with the problem of friction in an ad hoc way, and this report did not deal at all with complex contacts, such as contacts involving concavities. Complex contacts do not have a unique local normal, and so these contacts are similar to contacts with friction in the sense that the direction of applied force at a contact can fall within a range of possibilities.

The problem with friction and with complex contacts is that the grasp synthesis technique described in this report works best when matching a relatively small number of uniquely defined contact wrenches of a new grasp to the same small number of uniquely defined contact wrenches of a grasp prototype. Substituting wrench space regions for

these uniquely defined contact wrenches introduces substantial difficulties into this grasp synthesis algorithm.

Chapter 7 described some of the difficulties involved with one approach that attempted to transform wrench space regions into sets of unique contact wrenches by sampling the boundaries of these regions. This approach did not work well because it cluttered up the grasp prototype. It was difficult to find contact wrenches on the new target object to match each of the inflated number of prototype contact wrenches.

A better approach would be to find some way of streamlining the grasp prototype description as much as possible, keeping the number of contact wrenches used to describe that prototype to a minimum, while capturing most of the range of the prototype grasp. Complex contacts on the new target object could then be used in a flexible way to match one or more of the streamlined prototype wrenches. A twist on this would be to allow small sets of contacts on the new target object to match a single contact of the grasp prototype. All of these techniques introduce some additional complexity into the problem, but allow synthesis of a wider range of grasps.

8.1.3 Directional Tasks

Some tasks have non-zero components in some dimensions, but do not contain the wrench space origin in those dimensions (Figure 8.1). The task of drinking from a glass, assuming no arbitrary external wrenches are expected, falls into this class. Although the glass must be supported against gravity throughout the drinking motion, it does not matter if the grasp would let the glass fall if it were turned upside down. This type of task can be called a directional task, and it must be treated somewhat differently from tasks that do contain the wrench space origin in all dimensions in which they have non-zero components.

Figure 8.1 shows a two-dimensional example, including wrench space descriptions of both a grasp and a task. The main difference in generalizing such a grasp to fit the task is that the wrench space origin is included in the construction of the grasp space convex hull (Chapter 2). Contact constraint sets can be computed as described in Chapter 3, but when the grasp space is collapsed to form an intermediate convex hull tangent to the given task, then the facets of the grasp space convex hull that contain the wrench space origin must remain fixed to that wrench space origin (Figure 8.2). One interesting area of future work is to complete the description of the construction of contact constraint sets for some directional tasks and to use this construction to synthesize a range of grasps of new target objects.

8.1.4 Prototype Selection

The grasp synthesis algorithm described in this report requires a preprocessing step: somehow, a grasp prototype must be selected. Chapter 7 showed that a single grasp prototype, in particular the cylinder grasp prototype, can apply to a fairly wide range of target objects. This would tend to limit the size of the grasp library that would be required, and thus make the process of selecting an appropriate grasp prototype simpler. The question

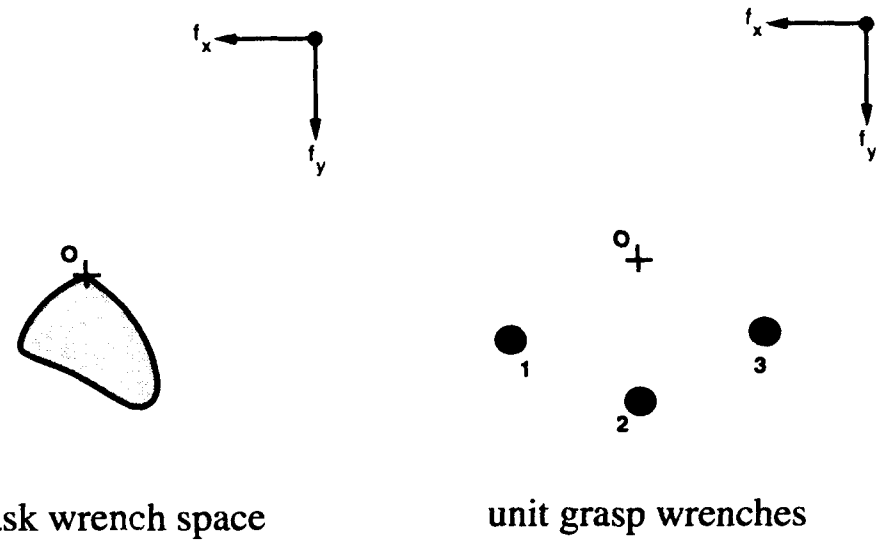


Figure 8.1: Wrench space of a directional grasp and a directional task. The grasp and task have two non-zero dimensions in which their wrench spaces do not contain the wrench space origin.

of how to perform such a selection in a principled way is difficult to answer, however. It would also be interesting to see if a grasp library could be constructed that would be guaranteed to cover a reasonably complete space of target objects and tasks.

8.1.5 Arm Constraints and Planning a Path

This report described a grasp synthesis technique that produced as output a space of collision-free hand configurations representing grasps of a new target object suitable for the intended task. A complete grasp plan, however, requires that a collision-free arm configuration be identified and a collision-free path be found from the current configuration of the robot into some high-quality, collision-free grasp. These are both hard problems, but the technique described in this report, which identifies a space of good hand configurations, provides a good starting point. This space of good hand configurations drastically constrains the space of good arm configurations and paths that may be found, but it also allows some flexibility in the final placement of the hand. This flexibility would not be present if only a single, optimal hand configuration had been identified.

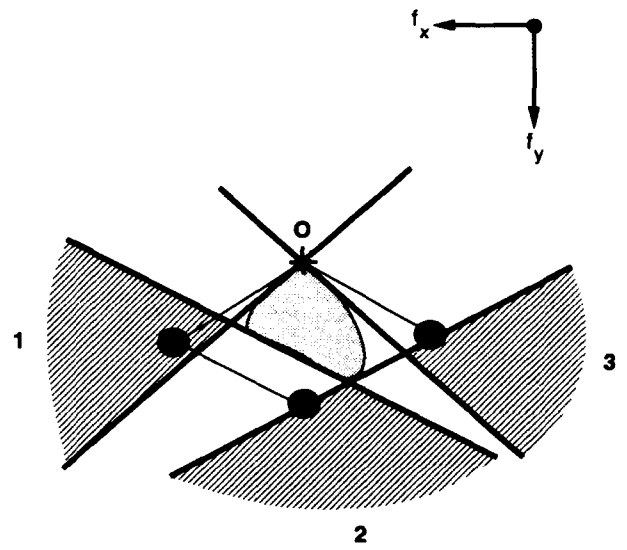


Figure 8.2: Contact constraint sets for a directional task. The wrench space origin is used as part of the construction.

Appendix A

Complete Algorithm Listing

This appendix presents a complete listing of all pseudocode functions presented or referenced in the body of the report. Functions are in alphabetical order by function name.

```
build_wrench(force, torque)
begin
   return(the wrench formed from the force and torque vectors);
end
```

```
collision_dist_of(local_contact_info) begin
   return(local_contact_info.collision_dist);
end
```

```
compute_cq_1 (orient, contact_assignment, prototype, hand_model, object, obstacles)
begin
   {unwind contact quality calculation for all links and link configurations}
   for link in links_of(hand_model) do
      for config in workspace_configs() do
         cq[link, config] :=
            compute_link_cq_1
               (link, config, orient, contact_assignment, prototype, hand_model,
                  object, obstacles);
   return(cq);
end


compute_cq_1_parallel
   (orient, contact_assignment, prototype, hand_model, object, obstacles)
begin
   {unwind contact quality calculation for all links and link configurations}
   for link in links_of(hand_model) do
      for i in processors() in parallel do
      begin
         link_config[i] := workspace_config(i);
         cq[link, i] :=
            compute_link_cq_1
               (link, link_config[i], orient, contact_assignment, prototype, hand_model,
                  object, obstacles);
      end
   {return parallel contact quality variable}
   return(cq);
end
```

```
compute_cq_2_parallel
   (orient, contact_assignment, prototype, hand_model, object, obstacles)
begin
   {unwind contact quality calculation for all links and link configurations}
   for link in links_of(hand_model) do
      for link_orient in workspace_orients() do
         for i in processors() in parallel do
         begin
            link_config[i] := workspace_config_2(link_orient, i);
            cq[link, link_orient, i] :=
               compute_link_cq_1
                  (link, link_config[i], orient, contact_assignment, prototype, hand_model,
                     object, obstacles);
         end
   {return parallel contact quality variable}
   return(cq);
end
```

```
compute_gq_1 (cq, hand_model)
begin
   {compute finger quality arrays fq for all fingers}
   for finger in fingers_of(hand_model) do
   begin
      prox_link := prox_link_of(finger, hand_model);
      fq[finger] := finger_quality_1(prox_link, cq, hand_model);
   end

   {compute grasp qualities gq for all wrist configurations}
   for wrist_config in workspace_configs() do
   begin
      {minimize grasp quality gq over finger qualities}
      gq[wrist_config] := MAX_QUALITY;
      for finger in fingers_of(hand_model) do
      begin
         prox_config := prox_link_config_of(finger, hand_model, wrist_config);
         if fq[finger, prox_config] < gq[wrist_config] then
            gq[wrist_config] := fq[finger, prox_config];
      end
   end
   return(gq);
end
```

```
compute_gq_1_parallel (cq, hand_model)
begin
   {compute parallel finger qualities fq for all fingers}
   for finger in fingers_of(hand_model) do
   begin
      prox_link := prox_link_of(finger, hand_model);
      fq[finger] := finger_quality_1_p(prox_link, cq, hand_model);
   end

   {compute grasp qualities gq over all processors}
   for i in processors() in parallel do
   begin
      wrist_config[i] := workspace_config(i);
      {minimize grasp quality gq over finger qualities}
      gq[i] := MAX_QUALITY;
      for finger in fingers_of(hand_model) do
      begin
         prox_config[i] := prox_link_config_of(finger, hand_model, wrist_config[i]);
         prox_proc[i] := proc_of(prox_config[i]);
         q[i] := fetch(fq[finger, prox_proc[i]]);
         if q[i] < gq[i] then
            gq[i] := q[i];
      end
   end
   return(gq);
end
```

```
compute_gq_2_parallel (cq, hand_model)
begin
   {compute parallel finger qualities fq for all fingers}
   for finger in fingers_of(hand_model) do
   begin
      prox_link := prox_link_of(finger, hand_model);
      fq[finger] := finger_quality_2_p(prox_link, cq, hand_model);
   end

   {compute grasp qualities gq over all processors}
   for wrist_orient in workspace_orients() do
      for i in processors() in parallel do
      begin
         wrist_config[i] := workspace_config_2(wrist_orient, i);
         {minimize grasp quality gq over finger qualities}
         gq[wrist_orient, i] := MAX_QUALITY;
         for finger in fingers_of(hand_model) do
         begin
            prox_config[i] := prox_link_config_of(finger, hand_model, wrist_config[i]);
            prox_orient := prox_orient_of(finger, hand_model, wrist_orient);
            prox_proc[i] := proc_of(prox_config[i]);
            q[i] := fetch(fq[finger, prox_orient, prox_proc[i]]);
            if q[i] < gq[wrist_orient, i] then
               gq[wrist_orient, i] := q[i];
         end
      end
   return(gq);
end
```

```
compute_gq_3_parallel
   (orient, contact_assignment, prototype, hand_model, local_contact_info)
begin
   {compute parallel finger qualities fq for all fingers}
   for finger in fingers_of(hand_model) do
   begin
      prox_link := prox_link_of(finger, hand_model);
      fq[finger] :=
         finger_quality_3_p
            (prox_link, orient, contact_assignment, prototype, hand_model,
               local_contact_info);
   end

   {compute grasp qualities gq over all processors}
   for wrist_orient in workspace_orients() do
      for i in processors() in parallel do
      begin
         wrist_config[i] := workspace_config_2(wrist_orient, i);
         {minimize grasp quality gq over finger qualities}
         gq[wrist_orient, i] := MAX_QUALITY;
         for finger in fingers_of(hand_model) do
         begin
            prox_config[i] := prox_link_config_of(finger, hand_model, wrist_config[i]);
            prox_orient := prox_orient_of(finger, hand_model, wrist_orient);
            prox_proc[i] := proc_of(prox_config[i]);
            q[i] := fetch(fq[finger, prox_orient, prox_proc[i]]);
            if q[i] < gq[wrist_orient, i] then
               gq[wrist_orient, i] := q[i];
         end
      end
   return(gq);
end
```

```
compute_gq_4_parallel
   (orient, contact_assignment, prototype, hand_model, local_contact_info)
begin
   {compute parallel finger qualities fq for all fingers}
   for finger in fingers_of(hand_model) do
   begin
      prox_link := prox_link_of(finger, hand_model);
      for wrist_orient in workspace_orients() do
      begin
         prox_orient := prox_orient_of(finger, hand_model, wrist_orient);
         fq[finger, wrist_orient] :=
            finger_quality_4_p
               (prox_link, prox_orient, orient, contact_assignment, prototype,
                  hand_model, local_contact_info);
      end
   end

   {compute grasp qualities gq over all processors}
   for wrist_orient in workspace_orients() do
      for i in processors() in parallel do
      begin
         wrist_config[i] := workspace_config_2(wrist_orient, i);
         {minimize grasp quality gq over finger qualities}
         gq[wrist_orient, i] := MAX_QUALITY;
         for finger in fingers_of(hand_model) do
         begin
            prox_config[i] := prox_link_config_of(finger, hand_model, wrist_config[i]);
            prox_proc[i] := proc_of(prox_config[i]);
            q[i] := fetch(fq[finger, wrist_orient, prox_proc[i]]);
            if q[i] < gq[wrist_orient, i] then
               gq[wrist_orient, i] := q[i];
         end
      end
   return(gq);
end
```

```
compute_link_cq_1
   (link, config, orient, contact_assignment, prototype, hand_model, object, obstacles)
begin
   cq := MIN_QUALITY;
   {link cannot be part of a good grasp if there is a collision}
   if no_collisions_1?(link, config, hand_model, object, obstacles) then
      {if link has no contact assignment, it is sufficient there be no collisions}
      if no_contact_assignment?(link, contact_assignment) then
         cq := MAX_QUALITY;
      else
         {if link has a contact assignment, it must make contact}
         if no_link_contact?(link, config, hand_model, object) then
            cq := 0;
         else
         begin
            {now we can compute contact quality}
            contact := contact_of(link, contact_assignment);
            wrench := contact_wrench(link, config, hand_model, object);
            cq := contact_quality(orient, contact, wrench, prototype);
         end
   return(cq);
end



compute_local_contact_info (object, obstacles)
begin
   local_contact_info.collision_dist :=
      {nearest distance to contact with one of the obstacles};
   local_contact_info.contact_dist :=
      {nearest distance to contact with the object};
   local_contact_info.contact_point :=
      {nearest point of contact with the object};
   local_contact_info.contact_feature :=
      {feature of object associated with the nearest point of contact};
   return({local_contact_info});
end
```

```
compute_local_cq
   (diam, collision_dist, contact_dist, contact_point, contact_feature, link, link_orient,
      orient, contact_assignment, prototype)
begin
   local_cq := MIN_QUALITY;
   if no_collisions_local?(diam, collision_dist, contact_dist) then
   begin
      if no_contact_assignment?(link, contact_assignment) then
         local_cq := MAX_QUALITY;
      else
         if no_contact_local?(diam, contact_dist) then
            local_cq := 0;
         else
         begin
            contact := contact_of(link, contact_assignment);
            wrench := contact_wrench_local(contact_point, contact_feature, link_orient);
            local_cq := contact_quality(orient, contact, wrench, prototype);
         end
   end
   return(local_cq);
end


contact_assignments_of(hand_model, prototype)
begin
   return(all assignments of contacts_of(prototype) to links_of(hand_model)
            such that there is exactly one link assigned to each contact);
end


contact_constraint_set_of(prototype, contact)
begin
   return(all vectors of the contact constraint set corresponding to contact of
            prototype);
end
```

```
contact_dist_of(local_contact_info)
begin
   return(local_contact_info.contact_dist);
end


contact_feature_of(local_contact_info)
begin
   return(local_contact_info.contact_feature);
end


contact_of(link, contact_assignment)
begin
   return(contact assigned to link);
end


contact_point_of(local_contact_info)
begin
   return(local_contact_info.contact_point);
end


contact_quality(orient, contact, wrench, prototype)
begin
   min_contact_quality := MAX_QUALITY;
   for vector in contact_constraint_set_of(prototype, contact) do
   begin
      tvector := transform_vector(vector, orient);
      cq := dot(tvector, wrench);
      if cq < min_contact_quality then
         min_contact_quality := cq;
   end
   return(min_contact_quality);
end
```

```
contacts_of(prototype)
begin
   return(indices to all contacts of prototype);
end


contact_wrench(link, config, hand_model, object)
begin
   return(some contact wrench from contact between link and object);
end


contact_wrench_local(contact_point, contact_feature, link_orient)
begin
   if is_edge?(contact_feature) then
      force := edge_inside_normal_of(contact_feature, link_orient);
   else
      {feature has a unique inside normal at the contact point}
      force := inside_normal_of(contact_feature, contact_point);

   dist := subtract(GRASP_CENTER, contact_point);
   torque := scale(cross(force,dist), TORQUE_MULTIPLIER);
   return(build_wrench(force, torque));
end


cross(vector1, vector2)
begin
   return(vector cross product of the two inputs);
end


diam_of (location, link, hand_model)
begin
   return({local diameter of link at the given location});
end
```

```
direction_of (step)
begin
   return({direction of motion in the processor grid to move from the
           location of this step to the location of the next step});
end


dot(vector1, vector2)
begin
   return(vector dot product of the two inputs);
end


edge_inside_normal_of(edge, link_orient)
begin
   return(the inside normal of edge, perpendicular to the link vector);
end
```

```
finger_quality_1(link, cq, hand_model)
begin
   {terminate at distal link}
   if no_next_link?(link, hand_model) then
      return(cq[link]);

   {get subchain quality for next link}
   next_link := next_link_of(link, hand_model);
   subchain_quality := finger_quality_1(next_link, cq, hand_model);

   {get subchain quality for current link}
   for config in workspace_configs() do
   begin
      {maximize next link subchain quality over joint angles}
      max_sq := MIN_QUALITY;
      for theta in joint_angles_of(link, hand_model) do
      begin
         next_config := next_link_config_of(theta, link, hand_model, config);
         if subchain_quality[next_config] > max_sq then
            max_sq := subchain_quality[next_config];
      end

      {take minimum of link and current subchain quality values}
      fq[config] := min(cq[link, config], max_sq);
   end
   return(fq);
end
```

```
finger_quality_1_p(link, cq, hand_model)
begin
   {terminate at distal link}
   if no_next_link?(link, hand_model) then
      return(cq[link]);

   {get subchain_quality for next link}
   next_link := next_link_of(link, hand_model);
   subchain_quality := finger_quality_1_p(next_link, cq, hand_model);

   {get subchain_quality for current link}
   for i in processors() in parallel do
   begin
      config[i] := workspace_config(i);
      {maximize subchain_quality over joint angles}
      max_sq[i] := MIN_QUALITY;
      for theta in joint_angles_of(link, hand_model) do
      begin
         next_config[i] := next_link_config_of(theta, link, hand_model, config[i]);
         next_proc[i] := proc_of(next_config[i]);
         sq[i] := fetch(subchain_quality[next_proc[i]]);
         if sq[i] > max_sq[i] then
            max_sq[i] := sq[i];
      end

      {take minimum of link and current subchain quality values}
      fq[i] := min(cq[link, i], max_sq[i]);
   end
   return(fq);
end
```

```
finger_quality_2_p(link, cq, hand_model)
begin
   {terminate at distal link}
   if no_next_link?(link, hand_model) then
      return(cq[link]);

   {get subchain_quality for next link}
   next_link := next_link_of(link, hand_model);
   subchain_quality := finger_quality_2_p(next_link, cq, hand_model);

   {get subchain_quality for current link}
   for link_orient in workspace_orients() do
      for i in processors() in parallel do
      begin
         config[i] := workspace_config_2(link_orient, i);
         {maximize subchain_quality over joint angles}
         max_sq[i] := MIN_QUALITY;
         for theta in joint_angles_of(link, hand_model) do
         begin
            next_config[i] := next_link_config_of(theta, link, hand_model, config[i]);
            next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
            next_proc[i] := proc_of(next_config[i]);
            sq[i] := fetch(subchain_quality[next_orient, next_proc[i]]);
            if sq[i] > max_sq[i] then
               max_sq[i] := sq[i];
         end

         {take minimum of link and current subchain quality values}
         fq[link_orient, i] := min(cq[link, link_orient, i], max_sq[i]);
      end
   return(fq);
end
```

```
finger_quality_2_p_better(link, cq, hand_model)
begin
   {terminate at distal link}
   if no_next_link?(link, hand_model) then
      return(cq[link]);

   {get subchain_quality for next link}
   next_link := next_link_of(link, hand_model);
   subchain_quality := finger_quality_2_p_better(next_link, cq, hand_model);

   {get subchain_quality for current link}
   for link_orient in workspace_orients() do
      for i in processors() in parallel do
      begin
         config[i] := workspace_config_2(link_orient, i);
         {maximize subchain_quality over joint angles}
         max_sq[i] := MIN_QUALITY;
         for theta in joint_angles_of(link, hand_model) do
         begin
            next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
            if subchain_quality[next_orient, i] > max_sq[i] then
               max_sq[i] := subchain_quality[next_orient, i];
         end

         {fetch maximum subchain quality value}
         next_pos[i] := next_link_pos_of(link, hand_model, config[i]);
         next_proc[i] := proc_of_2(next_pos[i]);
         sq[i] := fetch(max_sq[next_proc[i]]);

         {take minimum of link and current subchain quality values}
         fq[link_orient, i] := min(cq[link, link_orient, i], sq[i]);
      end
   return(fq);
end
```

```
finger_quality_3_p (link, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   {find maximal subchain quality}
   if no_next_link?(link, hand_model) then
      {subchain quality can be set to MAX_QUALITY}
      for link_orient in workspace_orients() do
         for i in processors() in parallel do
            max_sq[link_orient, i] := MAX_QUALITY;
   else
   begin
      {get subchain_quality for next link, all link orients}
      next_link := next_link_of(link, hand_model);
      subchain_quality :=
         finger_quality_3_p
            (next_link, orient, contact_assignment, prototype, hand_model, lc_info);

      {get subchain_quality for current link, all link orients}
      for link_orient in workspace_orients() do
         for i in processors() in parallel do
         begin
            {maximize subchain_quality over joint angles}
            max_sq[link_orient, i] := MIN_QUALITY;
            for theta in joint_angles_of(link, hand_model) do
            begin
               next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
               if subchain_quality[next_orient, i] > max_sq[link_orient, i] then
                  max_sq[link_orient, i] := subchain_quality[next_orient, i];
            end
         end
   end

   {collect link info along link axis, merge with subchain info for all link_orients}
   for link_orient in workspace_orients() do
      fq[link_orient] :=
         link_axis_quality_scan
            (max_sq[link_orient], link, link_orient, orient, contact_assignment,
               prototype, hand_model, lc_info);
   return(fq);
end
```

```
finger_quality_4_p
   (link, link_orient, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   {find maximal subchain quality}
   if no_next_link?(link, hand_model) then
      {subchain quality can be set to MAX_QUALITY}
      for i in processors() in parallel do
         max_sq[i] := MAX_QUALITY;
   else
   begin
      {maximize subchain_quality over joint angles}
      next_link := next_link_of(link, hand_model);
      for i in processors() in parallel do
         max_sq[i] := MIN_QUALITY;
      for theta in joint_angles_of(link, hand_model) do
      begin
         {get subchain_quality for current link and link_orient}
         next_orient := next_link_orient_of(theta, link, hand_model, link_orient);
         subchain_quality :=
            finger_quality_4_p (next_link, next_orient, orient, contact_assignment,
                                prototype, hand_model, lc_info);
         for i in processors() in parallel do
            if subchain_quality[i] > max_sq[i] then
               max_sq[i] := subchain_quality[i];
      end
   end

   {collect link info along link axis, merge with subchain info}
   fq := link_axis_quality_scan
            (max_sq, link, link_orient, orient, contact_assignment, prototype,
               hand_model, lc_info);
   return(fq);
end
```

```
fingers_of(hand_model)
begin
   return(unique indices to the fingers of hand_model);
end


hand_configs_of(hand_model)
begin
   return(all legal configs of the hand);
end


inside_normal_of(contact_feature, contact_point)
begin
   return(the inside normal of contact_feature at contact_point);
end


is_edge?(contact_feature)
begin
   if contact_feature is an edge then
      return(TRUE);
   else
      return(FALSE);
end


joint_angles_of(link, hand_model)
begin
   return(all 1D legal joint angles for the distal joint of link);
end
```

```
link_axis_quality_scan
   (max_sq, link, link_orient, orient, contact_assignment, prototype, hand_model, lc_info)
begin
   for i in processors() in parallel do
      subchain_cq[i] := min(0, max_sq[i]);

   for step in routing_path_of(link, link_orient, hand_model) do
   begin
      diam := diam_of(location_of(step), link, hand_model);
      for i in processors() in parallel do
      begin
         local_cq[i] :=
            compute_local_cq
               (diam, collision_dist_of(lc_info[i]), contact_dist_of(lc_info[i]),
                  contact_point_of(lc_info[i]), contact_feature_of(lc_info[i]), link,
                  link_orient, orient, contact_assignment, prototype);
         if local_cq[i] = MIN_QUALITY then
            subchain_cq[i] := MIN_QUALITY;
         if subchain_cq[i] > MIN_QUALITY then
            subchain_cq[i] := min(max(local_cq[i], subchain_cq[i]), max_sq[i]);
      end
      subchain_cq := local_fetch(subchain_cq, direction_of(step));
      max_sq := local_fetch(max_sq, direction_of(step));
   end
   return(subchain_cq);
end



link_config_of(link, hand_config)
begin
   return(config of link corresponding to hand_config);
end
```

```
link_contact?(link, config, hand_model, object)
begin
   if link in configuration config is in contact with object then
      return(TRUE);
   else
      return(FALSE);
end


link_of(contact, contact_assignment)
begin
   return(the link corresponding to contact of contact_assignment);
end


links_of(hand_model)
begin
   return(unique indices to links of hand_model);
end


location_of (step)
begin
   return({location of step of the routing path in the local link coord system});
end
```

```
lower_bound_gq_1(prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
      begin
         {compute contact quality array cq }
         cq := compute_cq_1
               (orient, contact_assignment, prototype, hand_model, object, obstacles);
         {compute grasp quality array gq}
         gq[orient, contact_assignment] := compute_gq_1 (cq, hand_model);
      end
   return(gq);
end
```

```
lower_bound_gq_1_parallel (prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
      begin
         {compute contact quality array cq }
         cq := compute_cq_1_parallel
               (orient, contact_assignment, prototype, hand_model, object, obstacles);
         {compute grasp quality array gq}
         gq[orient, contact_assignment] := compute_gq_1_parallel (cq, hand_model);
      end
   return(gq);
end
```

```
lower_bound_gq_2_parallel (prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
      begin
         {compute contact quality array cq }
         cq := compute_cq_2_parallel
               (orient, contact_assignment, prototype, hand_model, object, obstacles);
         {compute grasp quality array gq}
         gq[orient, contact_assignment] := compute_gq_2_parallel (cq, hand_model);
      end
   return(gq);
end


lower_bound_gq_3_parallel (prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
      begin
         {compute local collision and contact structure local_contact_info}
         local_contact_info := compute_local_contact_info(object, obstacles);
         {compute grasp quality array gq}
         gq[orient, contact_assignment] :=
            compute_gq_3_parallel
               (orient, contact_assignment, prototype, hand_model, local_contact_info);
      end
   return(gq);
end
```

```
lower_bound_gq_4_parallel (prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
      begin
         {compute local collision and contact structure local_contact_info}
         local_contact_info := compute_local_contact_info(object, obstacles);
         {compute grasp quality array gq}
         gq[orient, contact_assignment] :=
            compute_gq_4_parallel
               (orient, contact_assignment, prototype, hand_model, local_contact_info);
      end
   return(gq);
end
```

```
{compute entire space of lower bound gq measures based on cq measures}
lower_bound_gq_simple (prototype, hand_model, object, obstacles)
begin
   {align grasp prototype with target object}
   for orient in workspace_orients() do
      {assign links of the hand to contacts of the prototype}
      for contact_assignment in contact_assignments_of(hand_model, prototype) do
         {each hand config is a possible grasp}
         for hand_config in hand_configs_of(hand_model) do
         begin
            gq[orient, contact_assignment, hand_config] := MIN_QUALITY;
            if no_hand_collisions?(hand_config, hand_model, object, obstacles) then
            begin
               {lower bound gq is minimum cq over contacts}
               min_cq := MAX_QUALITY;
               for contact in contacts_of(prototype) do
               begin
                  link := link_of(contact, contact_assignment);
                  link_config := link_config_of(link, hand_config);
                  cq := MIN_QUALITY;
                  if link_contact?(link, link_config, hand_model, object) then
                  begin
                     wrench := contact_wrench(link, link_config, hand_model, object);
                     cq := contact_quality(orient, contact, wrench, prototype);
                  end
                  if cq < min_cq then
                     min_cq := cq;
               end
               gq[orient, contact_assignment, hand_config] := min_cq;
            end
         end
   return(gq);
end
```

```
next_link_config_of(theta, link, hand_model, link_config)
begin
   return(the config of next_link_of(link, hand_model) corresponding to
            link_config of link of hand_model and joint_angle theta);
end


next_link_of(link, hand_model)
begin
   return(an index to the distal neighbor of link link in hand_model);
end


next_link_orient_of(theta, link, hand_model, link_orient)
begin
   return(the orient of next_link_of(link, hand_model) corresponding to
            link_orient of link of hand_model and joint angle theta);
end


next_link_pos_of(link, hand_model, link_config)
begin
   return(the position of next_link_of(link, hand_model) corresponding to
            link_config of link of hand_model);
end


no_collisions_1?(link, config, hand_model, object, obstacles)
begin
   if link of hand_model lies in free space then
      return(TRUE);
   else
      return(FALSE);
end
```

```
no_collisions_local?(diam, collision_dist, contact_dist)
begin
   if (diam < (collision_dist - EPS_COLLISION)) and
      (diam < (contact_dist + EPS_CONTACT)) then
      return(TRUE);
   else
      return(FALSE);
end
```

```
no_contact_assignment?(link, contact_assignment)
begin
   if contact_of(link, contact_assignment) = NULL then
      return(TRUE);
   else
      return(FALSE);
end
```

```
no_contact_local?(diam, contact_dist)
begin
   if (diam < (contact_dist + EPS_CONTACT)) and
      (diam > (contact_dist - EPS_CONTACT)) then
      return(FALSE);
   else
      return(TRUE);
end
```

```
no_hand_collisions?(config, hand_model, object, obstacles)
begin
   if there is a collision between the hand of hand_model in
      configuration config and either object or obstacles then
      return(FALSE);
   else
      return(TRUE);
end
```

```
no_link_contact?(link, config, hand_model, object)
begin
   if link in configuration config is in contact with object then
      return(FALSE);
   else
      return(TRUE);
end


no_next_link?(link, hand_model)
begin
   if there is a link distal to link in hand_model then
      return(FALSE);
   else
      return(TRUE);
end


processors()
begin
   return(all available processors in the parallel machine);
end


proc_of(config)
begin
   return(processor in a configuration space processor grid closest to config);
end


proc_of_2(position)
begin
   return(processor in a position space processor grid closest to position);
end
```

```
prox_link_config_of(finger, hand_model, wrist_config)
begin
   return(the config of prox_link_of(finger, hand_model)
            corresponding to wrist configuration wrist_config);
end


prox_link_of(finger, hand_model)
begin
   return(an index to the proximal link of finger from hand_model);
end


prox_orient_of(finger, hand_model, wrist_orient)
begin
   return(the orient of the proximal link of finger corresponding to wrist_orient);
end


routing_path_of (link, link_orient, hand_model)
begin
   return({sequence of steps to move from the distal joint of link to the proximal
            joint of link given orientation link_orient .. each step consists of a
            location on the link in the local link coord system and the direction to
            move to get to the next location});
end


scale(vector, value)
begin
   return(vector scaled by value);
end


subtract(vector1, vector2)
begin
   return(vector difference of vector1 and vector2);
end
```

```
transform_vector(vector, orient)
begin
   return(vector resulting from the rotation of vector into the coordinate system
          represented by orient);
end


workspace_config(processor)
begin
   return(configuration represented by processor on a configuration space processor grid);
end


workspace_config_2(orient, processor)
begin
   return(configuration represented by the combination of orient and processor
          on a position space processor grid);
end


workspace_configs()
begin
   return(all configs in the workspace);
end


workspace_orients()
begin
   return(all orients in the workspace);
end
```

Bibliography

[1] Michael A. Arbib, Thea Iberall, and Damian Lyons. Schemas that integrate vision and touch for hand control. In Michael A. Arbib and Allen R. Hanson, editors, *Vision, Brain, and Cooperative Computation.* MIT Press, Cambridge, 1987.

[2] B. S. Baker, S. J. Fortune, and E. H. Grosse. Stable prehansion with a multi-fingered hand. In *Proc. IEEE Intl. Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[3] Christian Bard and Jocelyne Troccaz. Automatic preshaping for a dextrous hand from a simple description of objects. In *Proc. IEEE Intl. Workshop on Intelligent Robots and Systems*, Tsuchiura, Ibaraki, Japan, July 1990.

[4] Andrew Blake and Michael Taylor. Planning planar grasps of smooth contours. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Atlanta, Georgia, May 1993.

[5] David L. Brock. *A Sensor Based Strategy for Automatic Robotic Grasping.* PhD thesis, MIT Artificial Intelligence Laboratory, 1993.

[6] R. C. Brost. Automatic grasp planning in the presence of uncertainty. In *Proc. IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 1986.

[7] Michael E. Caine. The design of shape from motion constraints. Technical Report AI-TR-1425, MIT Artificial Intelligence Laboratory, 1993.

[8] I-Ming Chen and Joel W. Burdick. Finding antipodal point grasps on irregularly shaped objects. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Nice, France, May 1992.

[9] A. Christiansen, M. Mason, and T. Mitchell. Learning reliable manipulation strategies without initial physical models. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990.

[10] M. R. Cutkosky. Mechanical properties for the grasp of a robotic hand. Technical Report CMU-RI-TR-84-24, Carnegie Mellon Robotics Institute, 1984.

[11] Mark R. Cutkosky and Robert D. Howe. Human grasp choice and robot grasp analysis. In S.T. Venkataraman and T. Iberall, editors, *Dextrous Robot Hands*. Springer-Verlag, New York, 1990.

[12] M. Erdmann, M. Mason, and G. Vanecek, Jr. Mechanical parts orienting: The case of a polyhedron on a table. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Sacremento, California, 1991.

[13] Bernard Faverjon and Jean Ponce. On computing two-finger force-closure grasps of curved 2d objects. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Sacramento, California, 1991.

[14] Carlo Ferrari and John Canny. Planning optimal grasps. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Nice, France, May 1992.

[15] K. Goldberg and M. Mason. Bayesian grasping. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990.

[16] R. Grupen and R. Weiss. Force domain models for multifingered grasp control. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Sacremento, California, April 1991.

[17] Hideo Hanafusa and Haruhiko Asada. Stable prehension by a robot hand with elastic fingers. In *Proc. 7th Int. Symp. Industrial Robots*, pages 361–368, Tokyo, October 1977.

[18] Kenneth Hunt. *Kinematic Geometry of Mechanisms*. Oxford University Press, 1978.

[19] Thea Iberall and Christine L. MacKenzie. Opposition space and human prehension. In S.T. Venkataraman and T. Iberall, editors, *Dextrous Robot Hands*. Springer-Verlag, New York, 1990.

[20] J. W. Jameson. *Analytic Techniques for Automated Grasps*. PhD thesis, Department of Mechanical Engineering, Stanford University, January 1985.

[21] Marc Jeannerod. The timing of natural prehension movements. *Journal of Motor Behavior*, 16(3), 1984.

[22] Joseph L. Jones and Tomás Lozano-Pérez. Planning two-fingered grasps for pick-and-place operations on polyhedra. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990.

[23] J. Kerr and B. Roth. Analysis of multifingered hands. *The International Journal of Robotics Research*, 4(4), Winter 1986.

[24] D. G. Kirkpatrick, B. Mishra, and C. K. Yap. Quantitative steinitz's theorems with applications to multifingered grasping. In *Proc. 20th ACM Symposium on Theory of Computing*, Baltimore, Maryland, May 1990.

[25] Roberta Klatzky and Susan Lederman. Intelligent exploration by the human hand. In S.T. Venkataraman and T. Iberall, editors, *Dextrous Robot Hands*. Springer-Verlag, New York, 1990.

[26] C. Laugier, A. Ijel, and J. Troccaz. Combining vision based information and partial geometric models in automatic grasping. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990.

[27] Z. Li, P. Hsu, and S. Sastry. Grasping and coordinated manipulation by a multifingered robot hand. *The International Journal of Robotics Research*, 8(4), August 1989.

[28] Z. Li and S. Sastry. Task-oriented optimal grasping by multifingered robot hands. *IEEE Journal of Robotics and Automation*, 4(1), February 1988.

[29] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, and Patrick A. O'Donnell. *HANDEY A Robot Task Planner*. MIT Press, Cambridge, MA, 1992.

[30] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, Patrick A. O'Donnell, W. Eric L. Grimson, Pierre Tournassoud, and Alain Lanusse. Handey: A task-level robot system. In *4th Intl. Symp. on Robotics Research*, Santa Cruz, CA, 1987.

[31] Xanthippi Markenscoff and Christos H. Papadimitriou. Optimum grip of a polygon. *The International Journal of Robotics Research*, 8(2), April 1989.

[32] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4), 1987.

[33] J. Napier. The prehensile movements of the human hand. *Journal of Bone and Joint Surgery*, 38B(4), 1956.

[34] Thang N. Nguyen and Harry E. Stephanou. A computational model of prehensility and its applications to dextrous manipulation. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Sacramento, California, April 1991.

[35] Van-Duc Nguyen. The synthesis of stable force-closure grasps. Technical Report AI-TR-905, MIT Artificial Intelligence Laboratory, July 1986.

[36] I. Mazon P. Violero and M. Taix. Automatic planning of a grasp for a pick and place action. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990.

[37] Young C. Park and Gregory P. Starr. Grasp synthesis of polygonal objects using a three-fingered robot hand. *International Journal of Robotics Research*, 11(3), June 1992.

[38] Jocelyne Pertin-Troccaz. On-line automatic programming: A case study in grasping. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Raleigh, North Carolina, 1987.

[39] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Philadelphia, Pennsylvania, 1988.

[40] Nancy S. Pollard. The grasping problem: Toward task-level programming for an articulated hand. Master's thesis, MIT Artificial Intelligence Laboratory, May 1989.

[41] Jean Ponce, Steven Sullivan, Jean-Daniel Boissonnat, and Jean-Pierre Merlet. On characterizing and computing three- and four-finger force-closure grasps of polyhedral objects. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Atlanta, Georgia, 1993.

[42] J. K. Salisbury. *Kinematic and Force Analysis of Articulated Hands.* PhD thesis, Department of Mechanical Engineering, Stanford University, 1982.

[43] J. K. Salisbury and J. J. Craig. Articulated hands: Force control and kinematic issues. *The International Journal of Robotics Research*, 1(1), Spring 1982.

[44] Jeffery C. Trinkle. A quantitative test for form closure grasps. In *Proc. IEEE Intl. Conference on Intelligent Robots and Systems*, Raleigh, NC, July 1992.

[45] M. I. Vuskovic and A. K. Marjanski. Programmed synergy in dextrous robotic hands. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Atlanta, Georgia, May 1993.