HANDEY

A Robot Task Planner

Tomás Lozano-Pérez
Joseph L. Jones
Emmanuel Mazer
Patrick A. O'Donnell

The MIT Press
Cambridge, Massachusetts
London, England

This book was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Handey : a robot task planner / Tomás Lozano-Pérez . . . [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 0-262-12172-7
1. Robots—Programming. I. Lozano-Pérez, Tomás.
TJ211.45.H36 1992
629.8'92—dc20 92-7119
CIP

Contents

Series Foreword

Artificial intelligence is the study of intelligence using the ideas and methods of computation. Unfortunately a definition of intelligence seems impossible at the moment because intelligence appears to be an amalgam of so many information-processing and information-representation abilities.

Of course psychology, philosophy, linguistics, and related disciplines offer various perspectives and methodologies for studying intelligence. For the most part, however, the theories proposed in these fields are too incomplete and too vaguely stated to be realized in computational terms. Something more is needed, even though valuable ideas, relationships, and constraints can be gleaned from traditional studies of what are, after all, impressive existence proofs that intelligence is in fact possible.

Artificial intelligence offers a new perspective and a new methodology. Its central goal is to make computers intelligent, both to make them more useful and to understand the principles that make intelligence possible. That intelligent computers will be extremely useful is obvious. The more profound point is that artificial intelligence aims to understand intelligence using the ideas and methods of computation, thus offering a radically new and different basis for theory formation. Most of the people doing work in artificial intelligence believe that these theories will apply to any intelligent information processor, whether biological or solid state.

There are side effects that deserve attention, too. Any program that will successfully model even a small part of intelligence will be inherently massive and complex. Consequently artificial intelligence continually confronts the limits of computer-science technology. The problems encountered have been hard enough and interesting enough to seduce artificial intelligence people into working on them with enthusiasm. It is natural, then, that there has been a steady flow of ideas from artificial intelligence to computer science, and the flow shows no signs of abating.

The purpose of this series in artificial intelligence is to provide people in many areas, both professionals and students, with timely, detailed information about what is happening on the frontiers in research centers all over the world.

J. Michael Brady
Daniel G. Bobrow
Randall Davis

Preface

Research in robotics and artificial intelligence tends to go through cycles. For a few years the emphasis seems to be exclusively on methods targeted at narrow problems followed by a few years of attempts at integrating these methods into systems targeted at realistic tasks. It's easy to see why this pattern of development is useful. Building a system serves as a trial-by-fire for existing methods and points out holes in accepted problem definitions. On the other hand, the difficulties of building a system limit the type of innovations in approach that are practical in the component methodologies.

This book reports the results from the system-building part of our personal research cycle. In HANDEY we have tried to bring together several state-of-the-art technologies for developing model-based robot planning systems. Needless to say, we could not address all, or even most, of the problems in building a general-purpose task-level robot system. Nevertheless, we feel that our attempts shed some light on the state of the art in robot planning and may give some hints for where future work should be directed.

This book is meant to be read profitably by experts in robotics, but we have also attempted to make the book self-contained so that it remains accessible to students and robot users. This book, however, was not designed as a text or as a comprehensive review and evaluation of a field. Although we have included a brief review of some of the most relevant literature, we have relied heavily on referring the reader to other sources for more thorough reviews.

This book is organized as 8 chapters, as follows:

Chapter 1 presents an introduction to the HANDEY system and to task-level robot programming systems in general. It includes an example that aims to elucidate some of the reasons for the difficulty of robot programming.

Chapter 2 addresses the problem of planning pick-and-place tasks, which is HANDEY's target domain. It briefly outlines the structure of HANDEY and discusses some of the relevant literature.

Chapter 3 reviews some material in the areas of geometric modeling and kinematics required by the subsequent chapters. It also introduces the concept of configuration space, which plays a prominent role in HANDEY.

Chapter 4 presents the HANDEY gross-motion planner, which is responsible for planning motions for the robot arm between specified configurations. It presents two related approaches to gross-motion planning, one suitable for sequential computers and the other suitable for massively-parallel SIMD computers.

Chapter 5 takes up the problem of planning grasps in cluttered environments. It also presents two approaches to the problem, one based on a configuration-space representation and the other based on a potential-field model.

Chapter 6 considers the problem of planning regrasping operations.

Chapter 7 addresses the problem of coordinating two robots working in close proximity.

Chapter 8 reports some of our conclusions based on our experience in the development of the HANDEY system.

We want to thank the many people who at different times helped make the HANDEY system and this book possible. We would like to single out three people who contributed substantially to the initial development of HANDEY: Alain Lanusse, who developed the YASM geometric modeling system on which HANDEY is based, Pierre Tournassoud, who helped shape our approach to regrasping, and Eric Grimson, who helped develop our approach to object localization. In addition the following colleagues contributed through criticism, encouragement, and assistance: Mike Caine, Bruce Donald, Mike Erdmann, Matt Mason, Barb Moore, Sundar Narasimhan, Nancy Pollard, Jose Robles, and David Siegel.

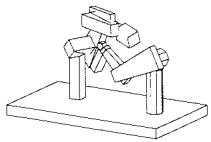
The work reported in this book was funded primarily by the Office of Naval Research under contracts N00014-85-K-0214 and N00014-86-K-0685. We would like to indicate our sincere appreciation to the Office of Naval Research for nearly a decade of stable support which brought this line of work from its infancy to its current state. We also owe our thanks to the ISTO office of DARPA, whose continued support of the Artificial Intelligence Laboratory at MIT has made it such a productive and exciting place to work. Additional support for the development of HANDEY was provided by an NSF Presidential Young Investigator Award to Lozano-Pérez, by the French CNRS, who partially supported Mazer's visits to MIT, and by the Digital Equipment Corporation whose support in the form of funds and robot hardware was very timely. We

also owe a debt of gratitude to the Systems Development Foundation whose support in the early eighties enabled the creation of the robotics group at the MIT Artificial Intelligence Laboratory.

Finally, we would like to thank our wives and our daughters who had to do without us far too often during the development of HANDEY and the completion of this book. Without their patience and support this book would never have seen the light of day.

Tomás Lozano-Pérez
Joseph L. Jones
Emmanuel Mazer
Patrick A. O'Donnell

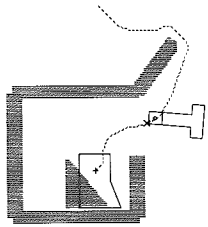
On the Margin



The margins of this book contain five **movies** that show operations planned by HANDEY. These movies can be animated by flipping the pages of the book. For the movies on the right margin, time moves along with increasing page number; for the movies on the left margin, time moves along with decreasing page number (starting at page 200).

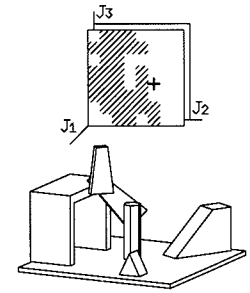
The movies help document some of the performance of the planners as well as illustrate some of the basic ideas in more detail than is possible with a few static snapshots. Furthermore, we thought they would be fun.

The left margin (on the even-numbered pages) has two movies. Ranging from top to bottom of the page, they are as follows:

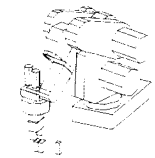


1. The **parallel motion-planner movie** shows a path planned by the parallel motion planner described in Section 4.5. The robot on the right is moving around the stationary robot on the left. The motion moves only the first three joints of the robot on the right.
2. The **potential-field movie** shows the motion of a gripper in the grasp plane reaching into a box to grasp a part using the method described in Section 5.4.2. The target point (marked by an '×') which attracts the gripper moves along the path indicated by a thin filament. This enables the planner to avoid local minima in the potential field.

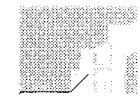
The right margin (on the odd-numbered pages) has three movies. Ranging from top to bottom of the page, they are as follows:

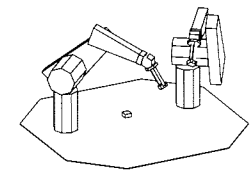


1. The **configuration space movie** shows a visual interpretation of the meaning of the configuration-space of the first three joints of a simple robot arm, with no joint limits. The location of the cross in the top diagram corresponds to the joint values of the robot in the three-dimensional model. The hatched regions indicate locations where there are collisions between the robot arm and its workspace. The three dimensional joint space of the arm is represented as a set of slices each representing the range of motions for the second and third joints; each slice is defined by a value of the first joint. This representation of the motion constraints on the arm underlies the implementation of the gross-motion planner.
2. The **regrasp movie** shows a pick-and-place operation that requires regrasping. The block on the far left needs to be grasped, rotated 90 degrees, and placed against the block on the right. To do this, HANDEY plans a grasp, a placement on the table (at a point further to the left), a new grasp, and then places the block at the destination.



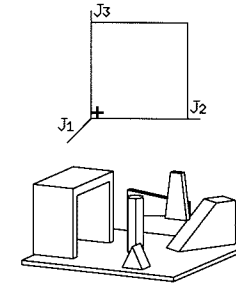
3. The **multi-arm coordination movie** shows two robots performing a sequence of operations in close proximity. The task is described in detail in Section 7.4.5. The **task completion diagram** for these tasks is also shown; the location on this diagram corresponding to each robot configuration is indicated. You can notice how each robot must wait for the other when it is about to enter a portion of the workspace which the other is occupying. (At a few points in the movie, the robots appear to pass through one another. In fact, one is passing behind the other, from the viewpoint of the "camera.")



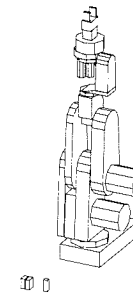


1 Introduction

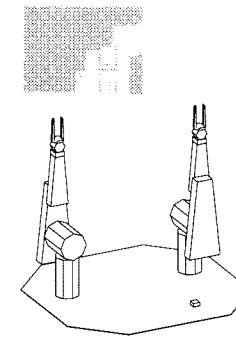
We are all aware that the science fiction depiction of robots as intelligent autonomous entities is far from the reality of today's industrial robots. Nevertheless, seeing an industrial robot in operation welding a car body or assembling a typewriter, it is difficult to resist the temptation of thinking of the particular operations being done as reflections of some more general competence in manipulation. We would like to think that a robot that could assemble part of a typewriter could readily assemble part of a bicycle or, even simpler, pick up a pen from a table. But, this is far from true. Each individual move of the robot during the typewriter assembly had to be carefully thought out by a programmer and carefully debugged over a period of months. Even small changes in the shape of the parts or their position could lead to disastrous results.



Surprisingly, the robot program to assemble a typewriter is not likely to have any subroutines that would be useful in assembling a bicycle or even a different model of typewriter. We will explore this point in more detail below, but the key observation is simple. The sequence of motions required to achieve a manipulation goal depends strongly on the detailed shape and arrangement of the parts and the dependence is more complex than can economically be encoded in simple subroutines. As a result, each new robot application is programmed starting from very primitive motion commands, such as "move to 1.2, 2.3, 3.4". This is analogous to programming a computer in machine code without even an assembler, and just as tedious and error prone.



If robots are to fulfill their scientific and economic potential, this programming bottleneck must be eliminated. This book describes a step in that direction. We set out to endow an industrial robot with the simplest competence one would naturally expect of a device built to manipulate objects. Namely, the ability to pick up user-specified objects and place them at user-specified positions. These manipulation problems are called **pick-and-place problems**. Note that pick-and-place is only a small subset of the manipulation problems we would ultimately expect a robot to perform. In particular, it does not include precise assembly, welding or grinding. But, almost every manipulation problem has some aspect of pick-and-place in it; they almost always involve grasping an object and taking it to a constrained position.



A robot system with pick-and-place competence could be commanded to move a part to a given position without describing in detail how this operation is to be done. In particular, the controlling computer would

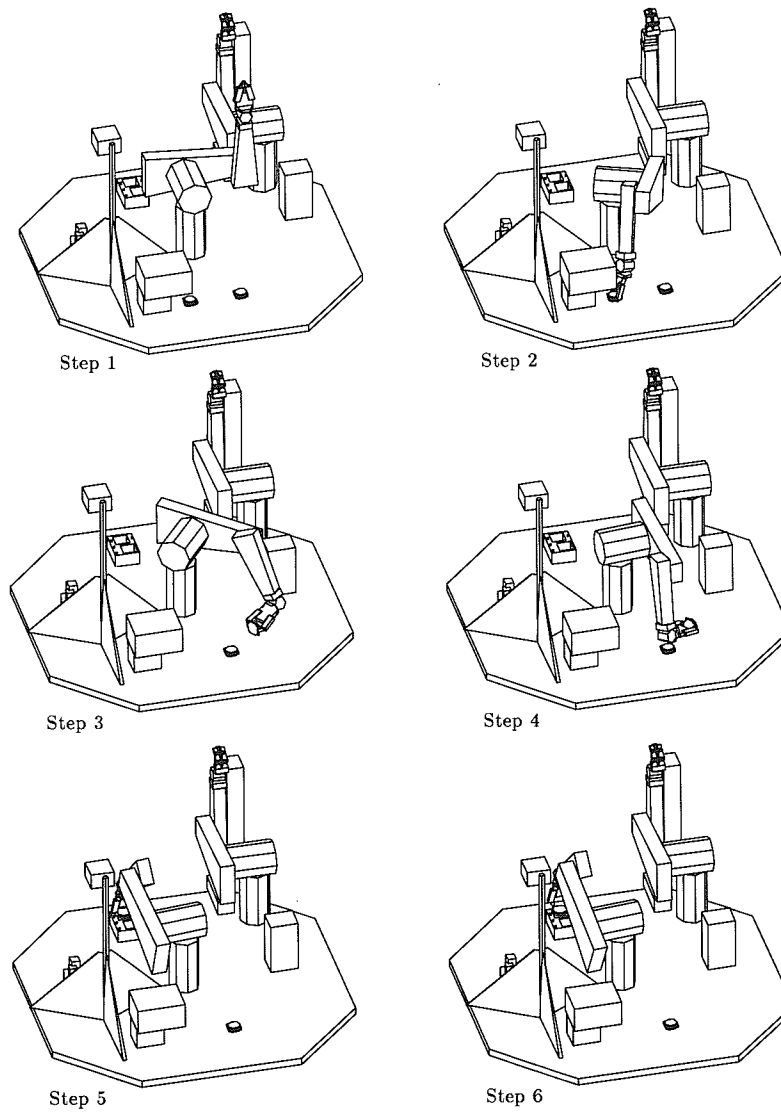
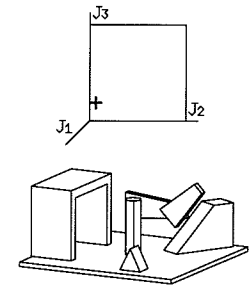
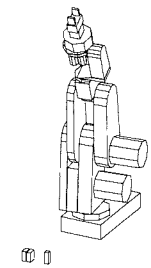


Figure 1.1
Steps in a pick-and-place operation planned by HANDEY.

select how the part is to be grasped and what path the robot arm should follow so as to reach the grasp while avoiding collisions with nearby objects. In other words, the controlling computer would compute all the detailed robot motions needed to achieve the goal. In this scenario, the user *would not* describe how to approach the part with the arm, how to grasp it or how to move it to its final destination. The user *would* provide a geometric description of the relevant parts and their positions relative to the robot. (This description may ultimately come from sensors but in the near term is likely to come from a graphics-based CAD system.) It would then be the job of the robot planner to determine the detailed operations necessary to perform the task.

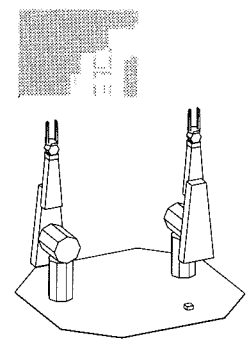


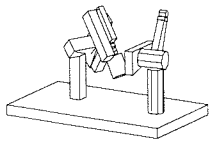
Of course, programming a pick-and-place task for a specific object, in a specific environment, with a specific robot, and to a specific destination is not that difficult. The difficulty resides in achieving some degree of generality. We want to have a system that can pick up any object, in any environment, with any robot, and take it to any destination. Of course, this level of complete generality is even beyond humans. We would settle for systems that can manipulate a wide class of objects in a wide class of environments using a wide class of robots. We designed our robot task planner, which we called HANDEY, with this level of capability in mind.



1.1 HANDEY

HANDEY is a robot programming system with a substantial level of pick-and-place competence. We call HANDEY a **task-level robot system** because it only requires a description of the task rather than a specification of the robot motions required to carry out the task. Given a goal to achieve and a polyhedral description of the environment and of the robot, HANDEY computes an appropriate sequence of robot commands to carry out the task. It can plan pick-and-place motions for objects modeled as general polyhedra, in accurately modeled environments, for robots with up to six joints, equipped with parallel-jaw grippers. For example, Figure 1.1 shows the key steps in a pick-and-place operation planned completely by HANDEY. The robot must pick up a part from below an overhanging box, flip it over and place it in its pallet location. Between Step 1 and Step 2, HANDEY plans a collision-free path from the initial position of the arm to a grasp. Both the choice of path and





the choice of grasp is carried out by HANDEY. No grasp was found to be both compatible with nearby objects and with the final position of the part and so HANDEY plans a regrasp. Step 3 shows the part being placed on the table on one of its stable states, chosen by HANDEY so as enable the next step, which is another grasp and another placement on the table, shown in Step 4. The net result of these steps is to flip the part over 180 degrees. At this point, with a feasible grasp available, a collision-free path is planned to take the part to its destination, shown in Steps 5 and 6.

The key property of a task-level system is that, as long as the task remains possible, one can make changes to the environment and one can expect that the system will generate new motions so as to achieve the task. For example, if one introduces new obstacles, HANDEY will automatically generate a new grasp and new motions compatible with the new environment. One can even change robots and expect the task to be performed without changing the specification.

HANDEY has been tested on numerous pick-and-place tasks, including parts ranging from wooden cubes to electric motors. The margin movies show that HANDEY has been been used to generate commands for different types of industrial robots; HANDEY has also been used to coordinate two arms working in the same workspace. HANDEY has also been tested with a module that locates the position of a specific part in a jumble of other parts. The system first builds a depth map of the jumble using structured light, locates edges in the depth map, and matches those edges to the model edges. HANDEY then plans how to grasp the part while avoiding the blocked areas in the depth map (see Section 5.3).

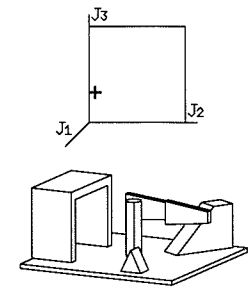
1.2 Robot programming

Task-level robot systems represent a substantial advance over traditional robot programming systems. In this section we review briefly some of these programming methods so as to motivate the need for task-level programming. For more detailed reviews, see [10, 45].

Robot programming today commonly uses three basic methods:

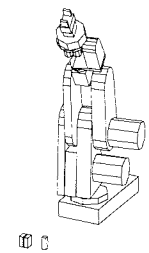
1. **Record/Playback** — The robot is led manually through a series of motions while recording key positions. This recorded sequence is then "played back" to repeat the motions.

2. **Textual programming** — The task is specified by writing a computer program that issues a sequence of position commands to the robot's control system.
3. **Off-line (graphical) programming** — A combination of textual programming and record/playback carried out on a simulated robot via an interactive graphical interface.



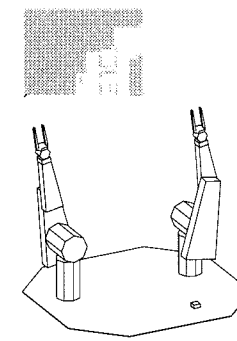
It is increasingly common to use combinations of these methods. Typically, a program is written embodying the sequence of operations to be performed and encoding any desired tests. Then, the robot is led manually (or graphically) to each of the positions specified in the textual program. The reasons for this mixture reflect the strengths and weaknesses of the methods.

The major weaknesses of record/playback are its inability to deal with variations in the task—such as picking up a part from a conveyor belt when the part position has been determined on-line by a vision system, and the difficulty of specifying tasks that are repetitive, but not completely identical on each repetition—such as picking parts from a rectangular pallet. Another limitation of record/playback is the difficulty of modifying an existing program to accommodate a modification to the task, for example, for a new automobile style.



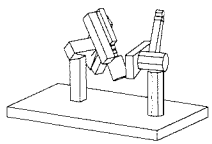
On the other hand, while textual programming can readily provide the flexibility missing from record/playback, it places the programmer in the impossible situation of trying to *imagine* what the motion generated by a set of numbers is going to be.

The combination of methods, as in off-line graphical systems, can reduce some of the weaknesses of the component methods. But, ultimately, all the traditional robot programming methods, individually and in concert, are limited by the difficulty of constructing higher-level procedures that can be used in more than one task. Let us explore this crucial issue in more detail.



1.3 Why is robot programming difficult?

The difficulty of programming a robot with current software tools only becomes apparent when one tackles a real problem. Therefore, we will consider a simple example that illustrates some of these difficulties.



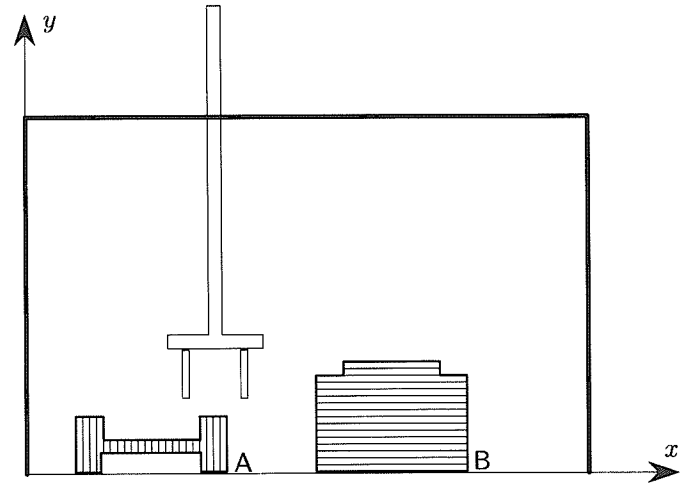


Figure 1.2
Simple robot and setup

1.3.1 A simple pick-and-place program

Consider the simple task of stacking two parts, called A and B (see Figure 1.2). We assume that the robot is capable of moving in the x and y directions and that it can open and close its gripper.

Of course, if parts A and B are always in the same place, we could use record/playback to program a sequence. But, if the actual positions are at all different from what they were during "recording" a catastrophic failure would ensue. Let us assume instead that part B is always fixed, it is a fixture, but part A is located by a vision sensor. We now want to write a program to pick up A and place it on B. This is a typical pick-and-place task.

The simple program in Figure 1.3 hides many of the actual details required in practice. For example, the position of part A would be represented by the coordinate transformation between the robot's coordinate frame and some standard frame attached to part A. The "grasp point on right of part A" denotes the composition of two coordinate transforms, one denoting a grasp position, *relative* to the standard frame for part A, composed with the transform denoting the actual position of part A.

```
begin
  locate part A
  move to the grasp point on right of part A
  close gripper
  move to the assembly position
end
```

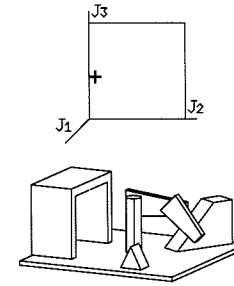
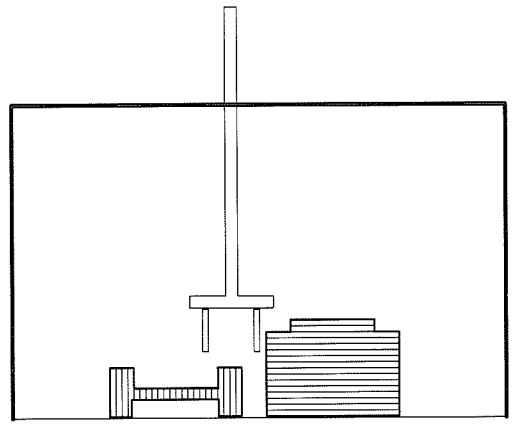


Figure 1.3
Program 1 — A simple pick and place program.



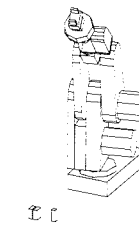
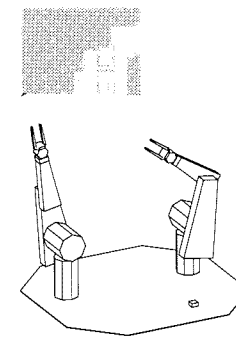


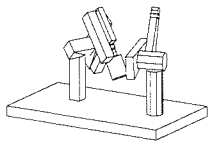
Figure 1.4
Grasp leading to a collision

The "assembly position" is also a coordinate expression denoting the position of the gripper when part A is at the final position. We will ignore these details in the rest of this example and simply ask you to keep in mind that these are barely outlines of the actual programs.



1.3.2 Grasping

What happens if part A is too close to part B, as in Figure 1.4? If we execute our simple program, a collision is possible during the robot's approach to the grasp point on A. We can avoid this problem by introducing a test into our program that results in choosing a grasp as



```
begin
  locate part A
  if part A is to the right of part B
    then set chosen grasp to grasp point on right of part A
    else set chosen grasp to grasp point on left of part A
  move to chosen grasp of part A
  close gripper
  move to assembly position
end
```

Figure 1.5
Program 2 – The grasp is chosen based on the relative positions of parts A and B.

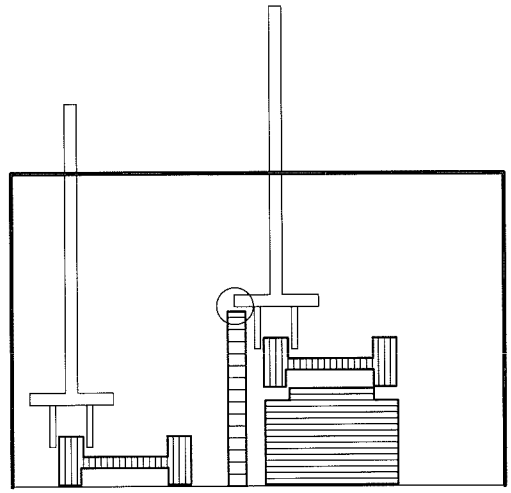
a function of the relative positions of A and B (see Figure 1.5). This example illustrates the main advantage of textual programming over the record/playback technique: it is possible to take actions based on the results of sensor measurements or other external data and computations.

Note that our simple strategy for choosing grasps can fail miserably, as shown in Figure 1.6. The problem is that when choosing a grasp we have to consider the environment at the destination as well as the environment around the current position of the grasped part.

1.3.3 Path planning

As a minimum, a robot program must guarantee that the robot moves without colliding with nearby objects. In the example in Figure 1.5, we saw the impact that this requirement has on the choice of grasp point. But, in general, any motion could cause a collision. For example, assume that our simple robot moves in a straight line from its starting position to the specified target position. Then the previous program will produce a collision as it is moving part A to the assembly position. To correct the program, we must specify intermediate points so as to avoid part B on the way to the destination. We should also consider the possibility of collision when approaching the chosen grasp point.

The path planning strategy embodied in the program in Figure 1.7 is very simple: the robot is first commanded to move "above" its destination and then commanded to move to the destination. While this



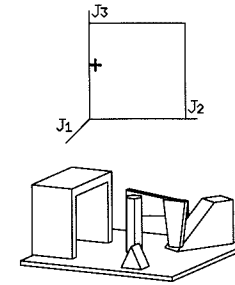
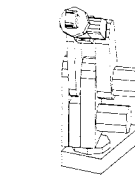


Figure 1.6
Grasp not suitable for the assembly



```
begin
  locate part A
  if part A is to the right of part B
    then set chosen grasp to grasp point on right of part A
    else set chosen grasp to grasp point on left of part A
  move to point 4 inches above chosen grasp of part A
  move to chosen grasp of part A
  close gripper
  move to point 4 inches above assembly position
  move to assembly position
end
```

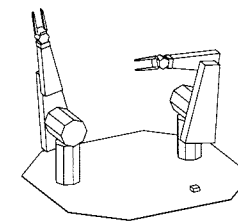
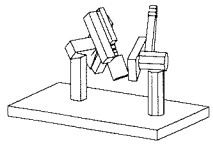


Figure 1.7
Program 3 – Each position is approached from above.



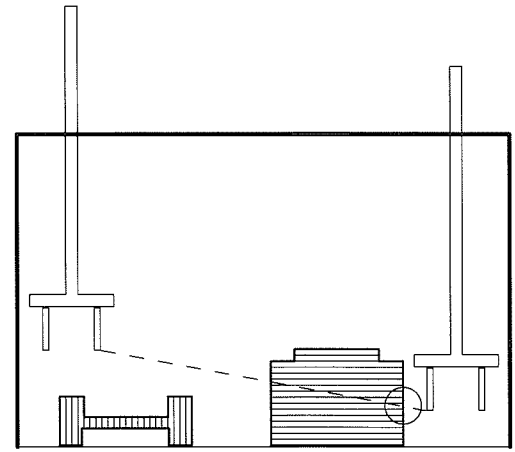


Figure 1.8
Collision with obstacles.

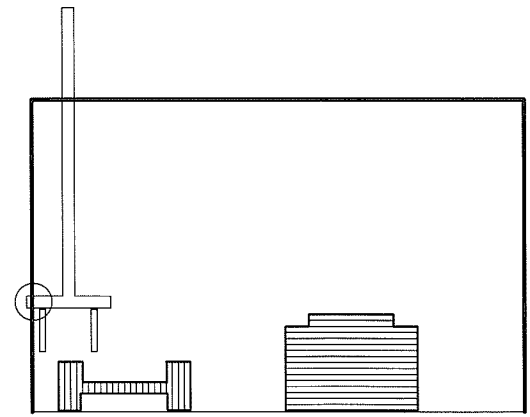
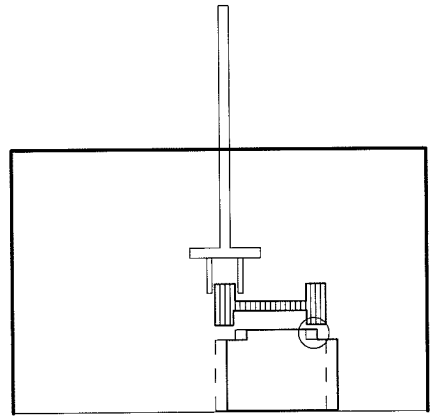


Figure 1.9
Exceeding the limits of travel.



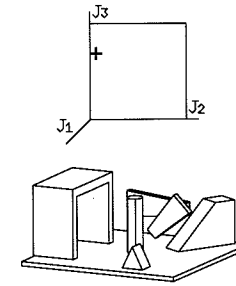
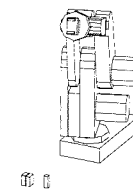


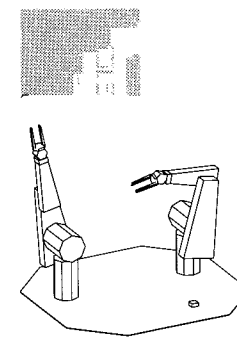
Figure 1.10
Collision due to uncertainty in position of part B relative to part A.

program will work in the situation shown in Figure 1.2, it will fail simply by changing the initial position of the robot, even without adding new obstacles (see Figure 1.8). Or, we can choose a position for part A which will make the grasp motion impossible due to limits in the legal motions (see Figure 1.9). Note that in these examples, the task is still achievable, either by introducing more intermediate points in the paths or by changing the chosen grasp.

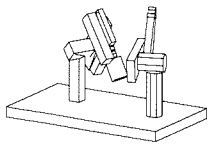


1.3.4 Uncertainty

In the previous examples, we have assumed that the position and shape of all the objects in the world, including the robot, are known perfectly and that the robot moves exactly as commanded. Neither of these assumptions is exactly true. Robots will deviate from their commanded positions by amounts that depend, at least, on the quality of the robot, the load being carried, and the speed of the motion. Furthermore, the knowledge of the positions and shapes of objects in the environment will not be exact, even if the position has been determined using some sensor. There will also be errors due to, at least, manufacturing tolerances and imperfections in the parts feeding and alignment mechanisms.



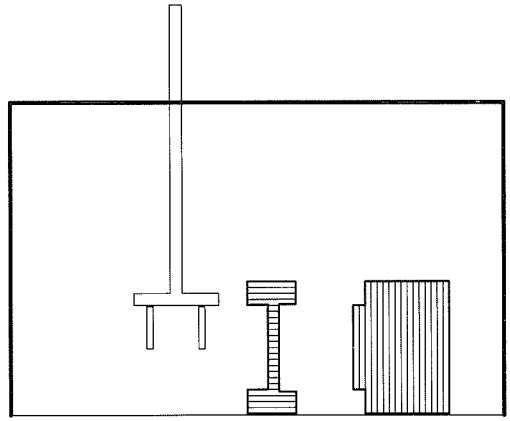
Ignoring these potential errors and uncertainties may lead to failures; Figure 1.10 shows a simple example. Ideally, we would like to construct



```
procedure Search
begin
  index = 0
  start y = current y
  while index * resolution < maximum error
    begin
      move in y to target y − 1 inch until touch
      if current y ≤ target y
      then return success
      else begin
        move in y to start y
        index = index + 1
        move in x to current x + (index * resolution)
      end
    end
  return failure
end
```

Figure 1.11
Search subroutine

a program that achieves its goal in spite of the presence of uncertainties. Sometimes this simply means exploiting the inherent physical and geometrical constraints of the task. For example, grasping a part tends to align it with the fingers; this can make up for a certain amount of uncertainty in the original position of the part. In other cases, sensors can be used to implement a strategy that overcomes some of the uncertainties. As a simple example, assume that our robot is equipped with a touch sensor that can detect contact between A and B and that the robot can stop quickly when the contact is detected. In that case, we can replace the last instruction in Program 3: "**move to** assembly position" by a call to the subroutine in Figure 1.11, which performs an exhaustive right-to-left search at a fixed resolution until it detects a successful assembly (by testing that the y coordinate of the position is close to the expected value). If the robot were equipped with a force-control capability, this simple search would better be implemented by a compliant motion [49, 52, 79].



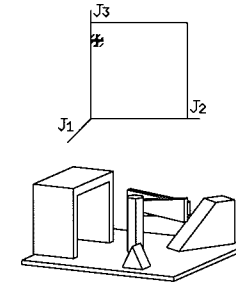
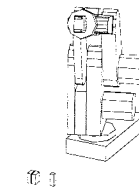


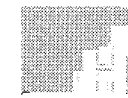
Figure 1.12
Same assembly - Different strategy

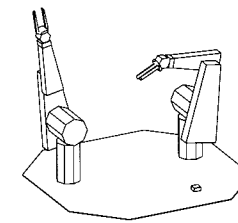
1.3.5 Error detection and recovery



In our small programs, we have made no provisions to detect abnormal situations. In the **Search** subroutine, for example, the presence of part B is not verified and part A is assumed to have been grasped properly. Checking for possible errors can account for the bulk of a real robot program. Unfortunately, trying to recover from these errors is very difficult since many possible causes could have been responsible for these errors. Even detecting success can be a challenging task [13, 15, 49].

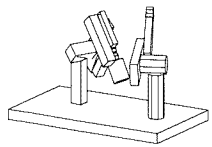
1.3.6 Discussion





The examples we have been considering are meant to show two important properties of robot programs:

- The motions required to achieve a task depend crucially on such factors as the shape of the parts, their relative position, and their positions relative to the robot. Relatively small changes in geometry can cause drastic changes in the strategy required for successful operation.
- The choice of motions cannot be made on purely local considerations. For example, the choice of initial grasp is influenced by the environment at the assembly position.



One example of a conceptually simple change with dramatic consequences on the structure of a robot program can be seen in Figure 1.12. The final assembly is the same one we have been considering, but rotated ninety degrees. The operations necessary to perform this rotated assembly, however, are radically different. It is *not* sufficient simply to rotate all the motions since the interactions of the parts with the table have changed. In particular, only one grasp is possible (and that is only possible if we allow the gripper to rotate 90 degrees) and the choice between alternative grasps is not relevant any more. Furthermore, the assembly of part A onto part B must now take into account the presence of the table, requiring a different strategy than that encoded in the **Search** subroutine.

The interdependence of decisions is best illustrated by the example in Figure 1.6. The presence of an obstacle at the assembly site affects the choice of initial grasp. Other interdependencies exist, as we will see in Chapter 2.

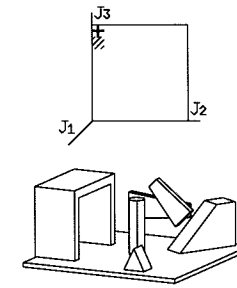
In our example, involving only two simple planar parts, it may be possible to write a complex procedure that tests for all relevant cases involving those two parts. But, in general, having to do that for each operation and any number of parts is out of the question. As a result, actual robot programs operate by sacrificing generality. The program is written to operate in a fixed environment with very few variations allowed. These programs have many "hidden assumptions," such as the expected initial position of the robot or the relative positions of the parts, that make them susceptible to failure in response to small changes in the environment and make them useless for other tasks.

Our goal in building HANDEY has been to develop a planning system that can take a description of a situation and generate a robot program that works for that particular situation. If the situation changes, a new robot program can be easily generated.

1.4 What HANDEY is and is not

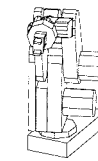
To understand HANDEY, it is important to clarify both what it does and what it does not do. Earlier in this chapter we described what HANDEY can do and most of this book is devoted to examining in detail

how HANDEY operates. In this section, we try to place HANDEY in the context of broad goals of robotics and, therefore, we will dwell primarily on what HANDEY does not do.

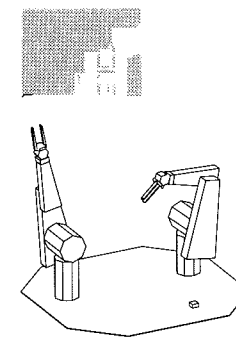


HANDEY is not intended to be a practical system used to program commercial robots. It is a research system, intended to help us evaluate the current state of the art in the technology for building task-level systems. Therefore, we emphasized ease of development over speed of execution. The system has not been subjected to the careful optimization and bug-fixing required for routine use. Nevertheless, HANDEY is reasonably fast. The planning time is usually less than 20 seconds[1] per pick-and-place step, counting each regrasping operation as a separate step.

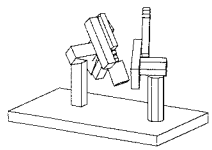
HANDEY takes as input a complete geometric and kinematic description of a robot and its environment and produces a program to achieve a user-specified pick-and-place operation in that environment. The programs produced by HANDEY are composed of explicit motion commands for the robot and its gripper. These commands specify the desired value for each of the robot's joints and desired gripper displacements. The assumption is that the robot controller will perform linear interpolation between successive commanded configurations of the robot and command the robot's position to follow that path. This is the most common mode of control for existing industrial robots.



The fact that HANDEY currently requires a complete environment model is not, in our view, a fundamental limitation. In fact, as we pointed out above, HANDEY has been tested in situations where range-sensing is used to locate some of the objects in the environment that is given as input to HANDEY. This could be done for the rest of the environment model. A more significant limitation is that HANDEY has no direct way of coping with variations in its environment *during* execution of its task. HANDEY does not generate robot programs that can sense the environment during execution of the pick-and-place operation and make decisions based on the sensing. HANDEY does not generate compliant motions [52] or guarded moves [45] to achieve robust operation or to detect failure. HANDEY does not even generate open-loop motions that can exploit task mechanics to produce the desired result in spite of initial uncertainty [7, 53].



[1] Running in Lucid Lisp on a Sun Sparc 2 workstation.



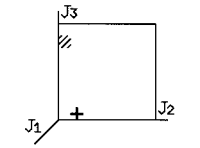
The fact that the current HANDEY does not use uncertainty-reduction techniques in its output programs does not mean that we are philosophically opposed to them. On the contrary! Neither does it reflect, we believe, an inherent limitation in approach. What it does reflect is a focus on the problem of deciding what to do in geometrically complex situations *after* one has determined the general structure of the world. Our key point is that deciding such apparently simple things as where to grasp a part involves rather complex interactions between task, environment and robot kinematics. Making these decisions is what HANDEY does. But, having made these decisions, HANDEY should proceed (but currently does not) to refine each step in the plan, such as grasping a part or placing it on a table, by further planning that takes into account the uncertainties in modeling and control [7, 13, 15, 49, 53, 79].

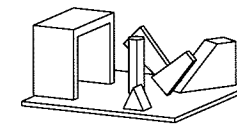
Having implemented the current HANDEY (several times now), it has become clear that one relatively simple type of uncertainty-reduction operation could fit readily within the existing HANDEY framework. In particular, approach, grasping and putdown motions could be generated as guarded moves. This would primarily require identifying the intended contacts for the motion during planning time and generating force tests to identify the first occurrence of these contacts and stop the motion. Our resources, however, have not been able to stretch to implement and test this approach.

Another limitation of the current HANDEY is that it does not plan the sequence of operations required to, for example, assemble or disassemble parts. The input to HANDEY is a sequence of task-level commands which may be generated by a human user or such as might be generated by one of the Artificial Intelligence assembly planners [60]. One key difficulty is that traditional AI planners assumed that one could succinctly characterize the conditions on the input world that would guarantee that a task-level operation would be successful. This is essentially impossible in any practical case and HANDEY can fail to find a plan for apparently reasonable inputs; sometimes because there is no answer but also because HANDEY's algorithms are not complete. More recent AI planning systems can more readily cope with operators that can fail, for example [81]. HANDEY could be a component of such planning system, as suggested in [14].

2 Planning Pick-and-Place Operations

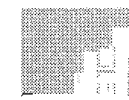
A **pick-and-place** operation is one where a user-specified object must be grasped at its current pose[1], called the **pickup pose**, and taken to a new user-specified position, called the **putdown pose**. The robot programming examples in Chapter 1 have illustrated many of the constraints that must be satisfied for a valid pick-and-place operation. The following list summarizes these constraints.

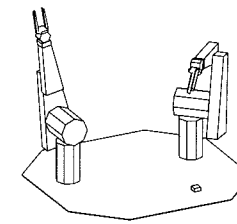




1. A collision-free, kinematically feasible[2] path exists from the robot's starting configuration[3] to the grasp at the object's pickup pose.
2. The grasp is stable—the grasped object is not free to twist or slip relative to the gripper.
3. This grasp is such that no part of the robot is in collision with any obstacle at either the object's pickup or putdown pose.
4. The grasp is kinematically feasible at *both* the object's pickup and putdown poses.
5. A collision-free, kinematically feasible path exists from the object's pickup to its putdown pose for the robot holding the object with the chosen grasp.
6. A collision-free, kinematically feasible path exists from the object's putdown pose to the robot's final configuration (the initial configuration for the next operation).

The key difficulty in satisfying all of these constraints on a pick-and-place operation arises from the interaction among the steps. In particular, the choice of a grasp brings together all the constraints; the grasp must be chosen so that no collisions arise and all the required paths are feasible. HANDEY attempts to deal in detail with most of these interacting constraints when choosing a grasp, for example, avoiding collisions and guaranteeing kinematic feasibility at both pickup and putdown poses. But, some other constraints are only handled approximately, for example, guaranteeing that the choice of a grasp does not interfere with finding collision-free paths for the remainder of the operation.





[1] A **pose** refers to the position and orientation of an object usually specified by a 4×4 homogeneous transformation matrix that relates a coordinate frame fixed on the object to a reference coordinate frame.

[2] A path is **kinematically feasible** if it is reachable by the robot.

[3] A **manipulator configuration** is most commonly specified by a vector of the manipulator's joint angles (see Chapter 3).

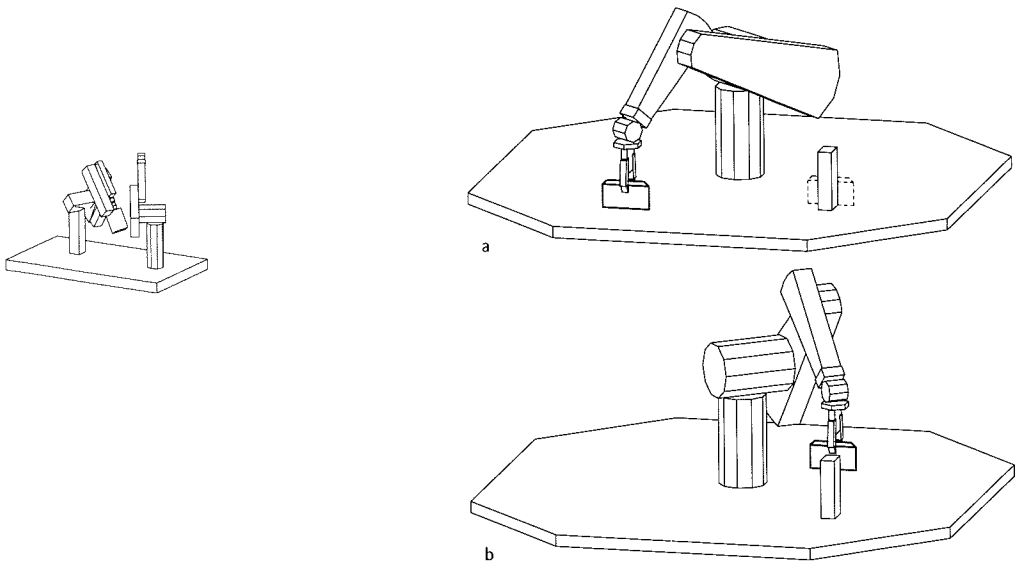


Figure 2.1
Interaction between stability of a grasp and collisions with obstacles

2.1 Examples of constraint interactions

A few examples will illustrate how a pick-and-place operation may fail if some constraint on the operation is satisfied without regard to the consequences for other parts of the operation.

Figure 2.1 demonstrates a simple interaction between the choice of a stable grasp and collisions with obstacles. In Figure 2.1(a) the goal is to move the highlighted object to the pose indicated in dashed lines. The grasp shown is one of the more stable possibilities. However, as shown in Figure 2.1(b), the grasp choice based only on stability of the grasp will cause the operation to fail – a less stable grasp must be used.

Another grasp choice that can lead to failure is one which is kinematically incompatible with the commanded putdown pose (see Figure 2.2). The goal in this case is simply to rotate the wedge shaped object about its lower left edge. However, it is impossible for the robot to carry out this motion if one makes the most obvious choice of initial grasp. People can solve both preceding problems by making the obvious grasp then

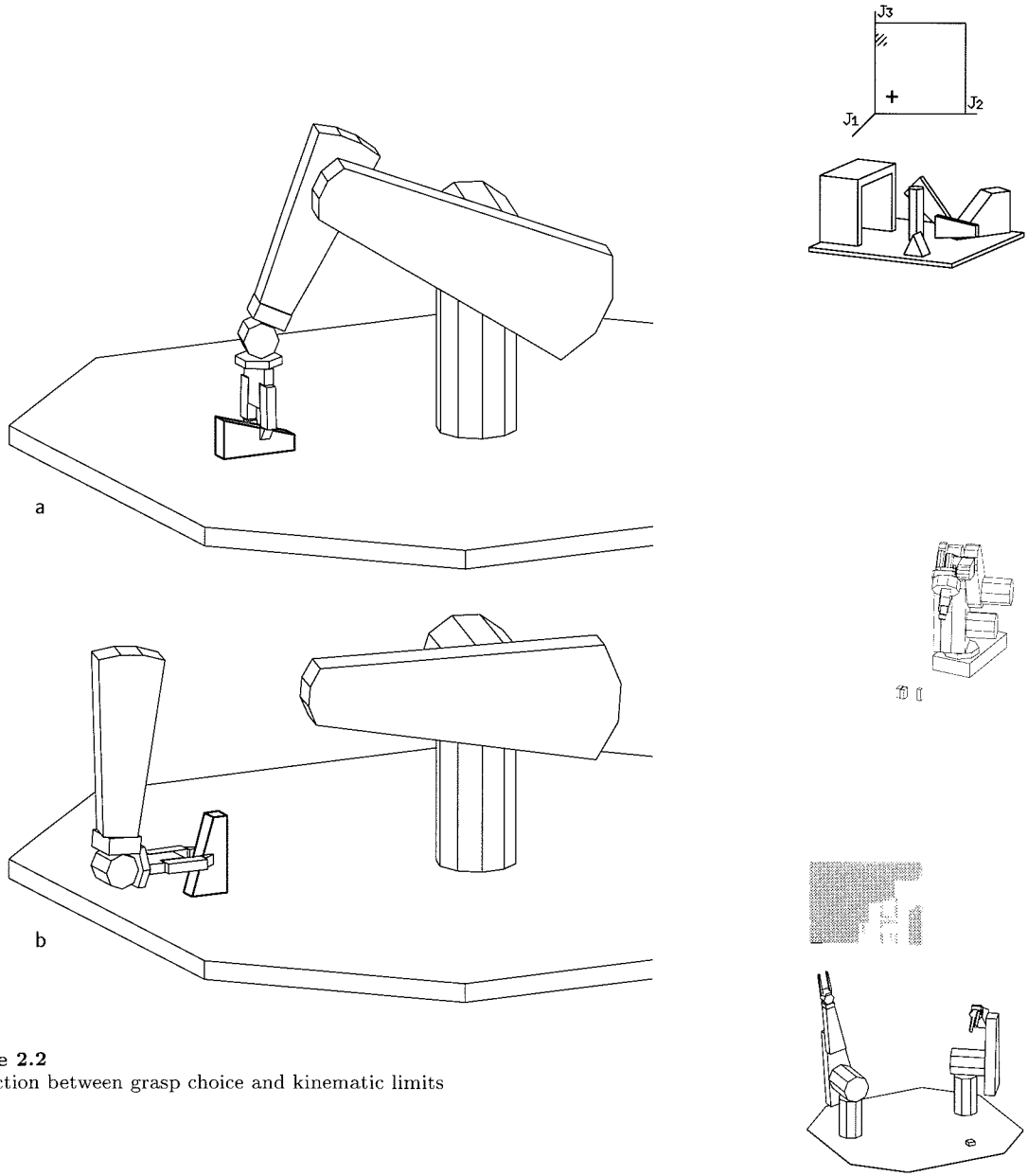


Figure 2.2
Interaction between grasp choice and kinematic limits

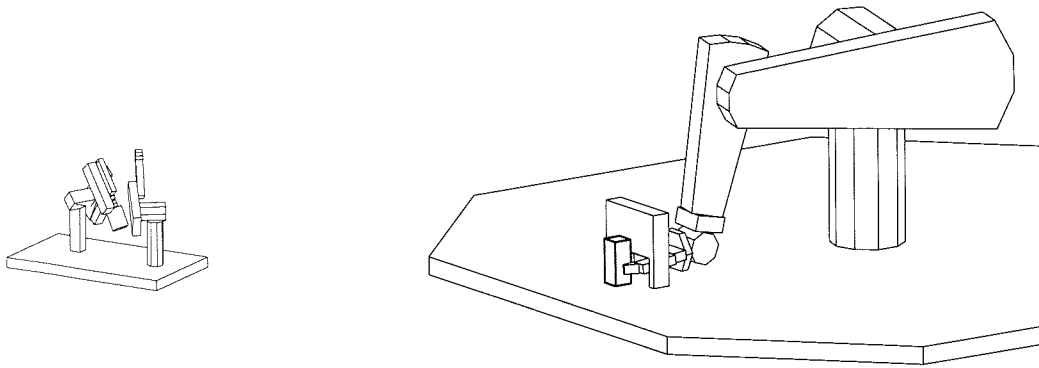


Figure 2.3
Interaction between grasp choice and pickup departure path

repositioning the part in their hands on the way to its putdown pose. A two fingered robot can achieve the same result only by planning a series of relatively slow and complicated regrasping operations (see Chapter 6).

Some more subtle interactions among the constraints are depicted in Figures 2.3, 2.4, and 2.5. Suppose that in Figure 2.3 the block is to be rotated about its lower left edge. The grasp shown satisfies the stability requirement, it produces no collisions with any obstacles, there is a kinematic solution at both the pickup an putdown poses, there is a viable path from the robot's starting location to the grasp, and a viable path departing the putdown location exists. However, as can readily be seen, for this grasp there is no collision-free path for the robot holding the object to reach the goal.

Figure 2.4 shows a T-shaped block which the robot has just placed in its putdown pose on the table. Even at this point the grasp choice can cause problems. In this example it is impossible for the robot to extricate itself without colliding either with the object it has just placed or the block behind the gripper.

It may appear in Figure 2.5 that the plan to place the T-shaped object on the large block has succeeded. In fact, once again the combination of grasp choice and kinematic solution at the putdown pose has caused a failure. In this case the shoulder (which connects the horizontal cylinder and the larger wedge shaped link) and the elbow joint (which connects

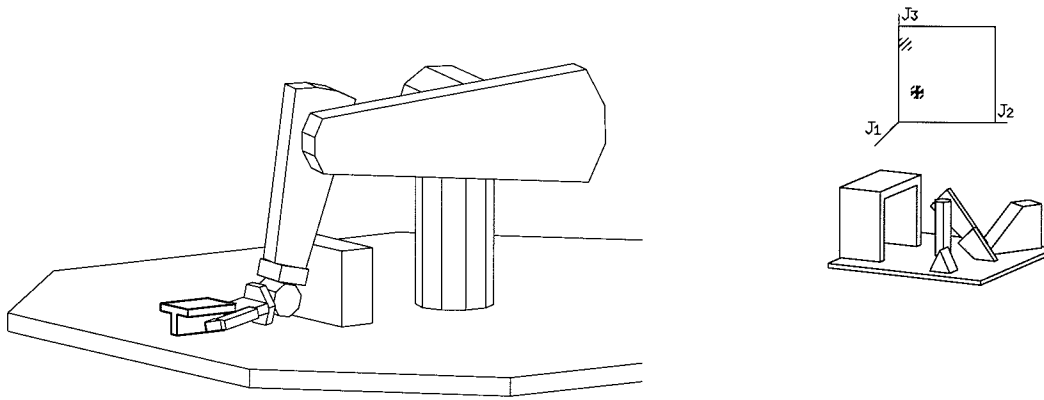


Figure 2.4
Interaction between grasp choice and putdown departure path collisions

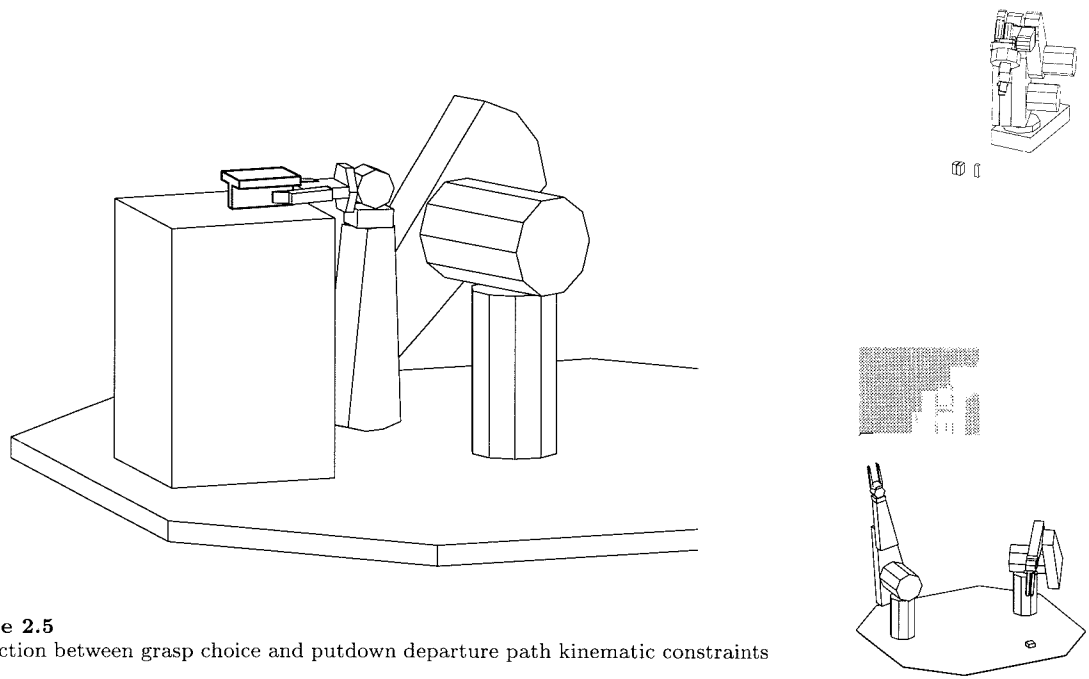
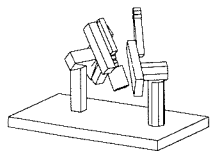


Figure 2.5
Interaction between grasp choice and putdown departure path kinematic constraints

the two wedge shaped links) are both so close to their kinematic limits that it is not possible for the gripper to back away from the grasp without disturbing the T-shaped part.

In each of the examples just given the following are true:

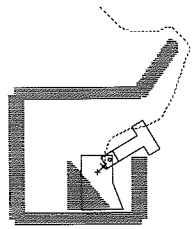


- The part to be moved is small and geometrically simple,
- The world is simple and sparsely populated, and
- The specified task is simple and does not require high precision.

However, even in these straightforward situations a pick-and-place plan which ignores constraint interaction can readily fail.

2.2 A brief overview of HANDEY

The key design objective for HANDEY has been to deal directly with as many of the constraint interactions illustrated above as possible, but within the context of a relatively modular design. This proved a bit tricky (see Chapter 8) but in the resulting design, the kernel of HANDEY is composed of four nearly independent planners:



- the gross-motion planner,
- the grasp planner,
- the regrasp planner, and
- the multi-arm coordinator.

HANDEY breaks down the planning of a pick-and-place operation into a number of steps. Each planner's responsibility is limited to a subset of these steps. At the start of a pick-and-place operation the robot will be in some initial configuration, Figure 2.6(a). The gross motion planner plans a motion that will take the robot from its initial configuration to an approach configuration chosen by the grasp planner Figure 2.6(b). A grasp configuration that is reachable by the robot at both pickup and putdown is chosen by the grasp planner, Figure 2.6(c) and Figure 2.6(d). The grasp planner also computes a path between the approach and grasp configurations. The gross motion planner then moves the grasped object to the putdown pose, Figure 2.6(d). The grasp planner chooses a departure configuration, Figure 2.6(e) and computes a path between the putdown grasp configuration and the departure configuration.

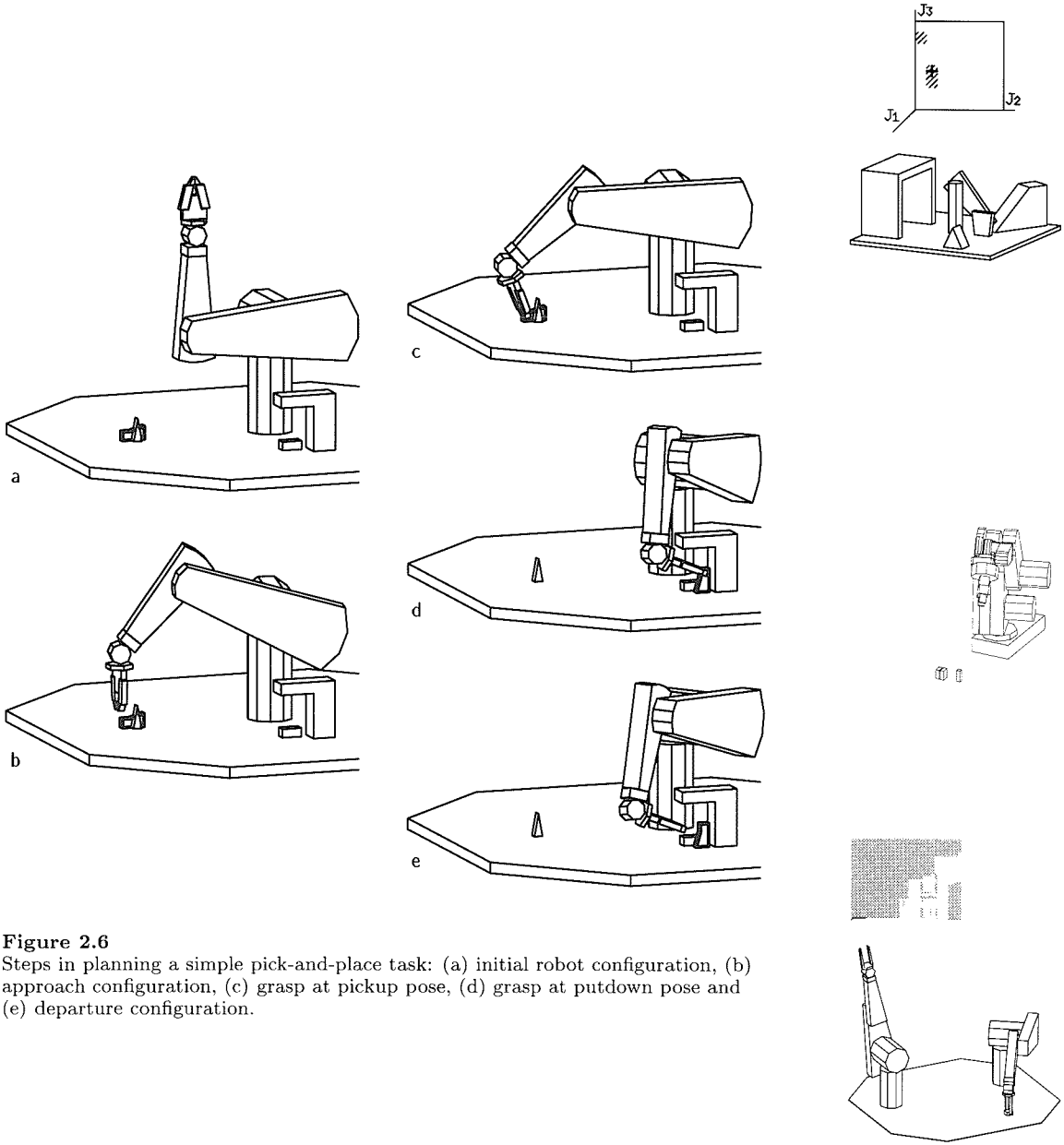
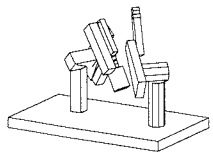


Figure 2.6
Steps in planning a simple pick-and-place task: (a) initial robot configuration, (b) approach configuration, (c) grasp at pickup pose, (d) grasp at putdown pose and (e) departure configuration.



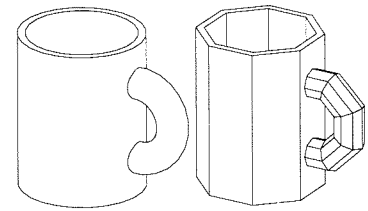
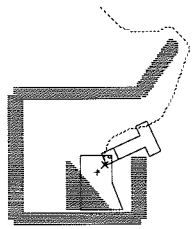


Figure 2.7
Objects are modeled as polyhedra.

If a grasp cannot be found that is consistent with both the part's pickup and putdown poses, the regrasp planner is called to generate a sequence of grasps and placements (at a fixed spot on the table) that will permit the robot to place the part at its intended destination. Each grasp and placement chosen by the regrasp planner will also require planning gross-motions, approach motions, and departure motions.

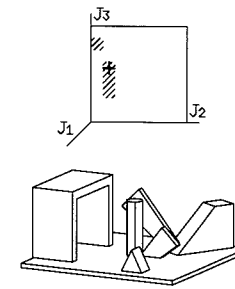


2.3 The HANDEY planners

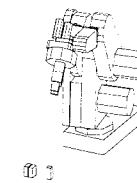
The HANDEY planners operate on five main data-structures:

- **Part**: A **part** is a three-dimensional geometric model of an object placed in a particular pose. The shape of the part is represented as a polyhedron. HANDEY is able to manipulate smooth objects, non-convex objects, and even make use (for grasping) of holes in objects. The only requirement is that a good polyhedral approximation, as illustrated in Figure 2.7, be available for each object to be manipulated (see Chapter 3).
- **World**: A **world** is a three-dimensional geometric model of a scene, containing one or more robots and multiple parts, at a given instant in time. Many of the figures in this book are displays of HANDEY worlds. To be legal, a world should not include a model of a robot colliding with any part or any other robot in the world.
- **Robot**: A **robot** describes the kinematic structure of an arm and contains the geometric model of its links. It also includes information about its gripper and an inverse kinematics function, that is, a function that can solve for the robot's joint angles given a coordinate frame specifying the pose of the gripper (see Chapter 3).

- **Goal**: A **goal** can either specify a set of joint angles for a given robot arm or the desired final pose of a part. The current version of HANDEY requires that at the goal pose there should not be any contact between the moved part and other objects in the world. This restriction is imposed because the current HANDEY places the part at the final pose using position control without contact sensing. If the final pose is too close to other objects, control and modeling errors will almost guarantee a collision.
- **Plan**: A **plan** describes the elementary robot and gripper motions produced by a planner to achieve the goal. It also contains the nominal world produced as the result of each operation. The **initial world** of a plan describes the nominal world when the plan is started. The **final world** of a plan describes the nominal world when the plan is completed. This final world will serve as the initial world of the next operation.



HANDEY planners are invoked using these data structures as arguments. In case of a success they return a plan data structure that can be used to simulate the plan or execute it on a real robot. When a planner fails to plan the desired operation a **failure** data structure is returned. This data structure contains a reason for failure and, optionally, a list of parts in the world which may prevent the planner from achieving its goal. Other planners can use this data-structure to construct new goals that may help achieve the original task.

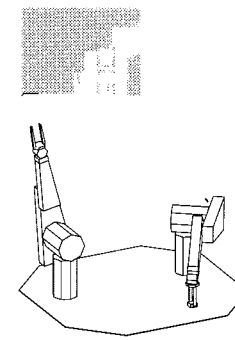


2.3.1 The gross motion planner

The gross motion planner is invoked as follows:

Move(*goal*, *robot*, *world*)

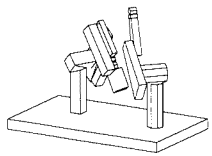
The gross-motion planner will generate a plan describing a collision-free path from the configuration of the robot in the specified world model to the configuration specified in the goal. A failure report is produced either when the goal is not kinematically feasible, when there is a collision at the goal, or when no path can be found between the initial and goal configurations of the robot. Chapter 4 describes the gross-motion planner.



2.3.2 The grasp planner

The grasp planner is invoked as follows:

Grasp(*goal*, *robot*, *world*, *depart*?)

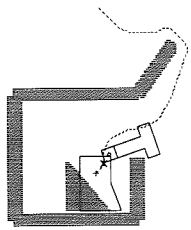


If the *depart*? flag is true, the grasp planner will generate two plans. The first one is to approach and grasp the part referred to in the goal at its pickup pose in the specified world. The second is to release the part at its putdown pose (the goal) and to back away slightly from the grasp. The grasp is chosen so that there are no static collisions at pickup or putdown and so that the approach and departure paths exist. A failure report will be issued if such a grasp does not exist. If the *depart*? flag is false, only the approach and grasp plan is computed. Chapter 5 describes the grasp planner.

2.3.3 The regrasp planner

The regrasp planner is invoked as follows:

Regrasp(*goal*, *robot*, *world*)



When this planner is called the robot must be holding the part which is specified by the goal. The planner will generate an appropriate sequence of pairs of robot operations consisting of placing the part on the table and regrasping it with another grasp until a grasp compatible with the putdown pose in the goal is found. Chapter 6 describes the regrasp planner.

2.3.4 The multi-arm coordinator

The multi-arm coordinator is invoked as follows:

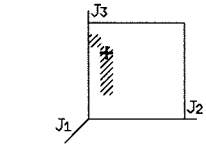
Coordinate($plan_1$, $plan_2$, *world*)

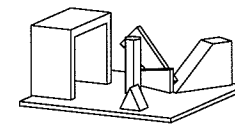
$plan_1$ and $plan_2$ are plans for two different arms. The multi-robot planner produces a combined plan for the two robot arms that will move the arms through the same paths but allows them to move at the same time whenever possible. Chapter 7 describes the multi-arm coordinator

2.4 Combining the planners

The planners described above are the bulk of the HANDEY system. The pick-and-place competence of the system arises from simple combina-

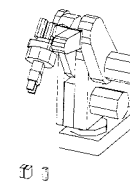
tions of these planners. For example, the simplest pick-and-place planner could be constructed as follows:

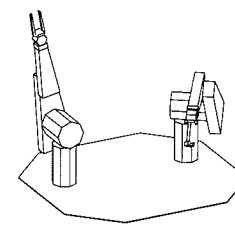




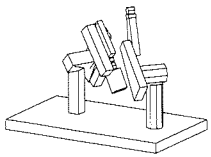
```
procedure Simple-Pick-And-Place(goal,robot,init-world)
begin
  if (app,dep=Grasp(goal,robot,init-world,true)) then
    if (gm-1=Move(first-world(app),robot,init-world)) then
      if (gm-2=Move(first-world(dep),robot,final-world(app))) then
        return (merge-plans (gm-1,app,gm-2,dep))
        else return (failure("can't move to putdown pose"))
      else return (failure("can't move to pickup pose"))
    if (app) then
      return (failure("can't find a pickup grasp/approach"))
      else return (failure("can't find a putdown grasp/departure"))
  end
```

Here we have assumed that a failure is interpreted as a boolean false by **if**. Note that the order in which the motions are planned is not the order in which they are executed. In particular, the grasp is planned first since it is the most constraining operation. The grasp planner computes both an approach (**app**) and a departure plan (**dep**). The approach motion moves from a point near the part's pickup pose (Figure 2.6(b), page 23) to the grasp (Figure 2.6(c)). The departure motion moves from the grasp at the putdown pose (Figure 2.6(d)) to a point a little away from the part (Figure 2.6(e)). The first gross motion (**gm-1**) moves from the initial configuration of the robot (Figure 2.6(a)) to the first configuration in the approach path (Figure 2.6(b)). The second gross motion (**gm-2**) moves from the last configuration in the approach path, that is, the pickup grasp (Figure 2.6(c)), to the first configuration in the departure path, that is, the putdown grasp (Figure 2.6(d)). The merged paths are then returned.

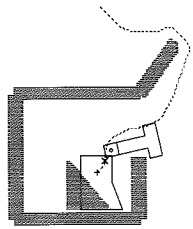




The actual program in the HANDEY system is not substantially different from this. In practice, this simple pick-and-place planner will fail for most combinations of pickup and putdown poses, primarily because of limitations in the range of joint motions of most robot manipulators. The grasps compatible with the pickup pose will typically not be compatible with the putdown pose. We can construct a much more robust pick-and-place planner by using the regrasp planner described above. The new planner is constructed as follows:



```
procedure Pick-And-Place(goal,robot,init-world) begin
  if (simple=Simple-Pick-And-Place(goal,robot,init-world))
    then return (simple)
  if (app=Grasp(null,robot,init-world,false)) then
    if (regrasp=Regrasp(goal,robot,final-world(app))) then
      if (gm-1=Move(first-world(app),robot,init-world)) then
        if (gm-2=Move(first-world(dep),robot,final-world(regrasp)))
          then return (merge-plans(gm-1,app,regrasp,gm-2))
          else return (failure("can't move to putdown pose"))
        else return (failure("can't move to pickup pose"))
      else return (failure("can't find a regrasp plan"))
    else return (failure("can't find a pickup grasp/approach"))
end
```

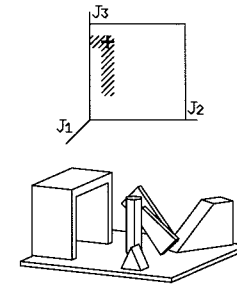


First, one tries to plan the motion using the simple planner above. If that plan (**simple**) fails, a new attempt is made using the regrasp planner. First, the grasp planner is used to plan an approach to a grasp (**app**), ignoring the putdown pose. Then the regrasp planner plans a sequence of grasps and placements (**regrasp**) that leave the part in a grasp suitable for the putdown. Then the remaining gross motions are planned, the first (**gm-1**) to approach the initial grasp (**app**) and the second (**gm-2**) to place the part at the putdown pose.

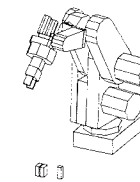
In practice, the reason for the failure of the simple planner would be checked to make sure that the failure is one that can be overcome by a regrasping operation. The Right-Margin Movie 2 shows the robot motions produced by this pick-and-place planner, including a regrasping operation.

Many variations and extensions of these pick-and-place planners are possible. For example, the initial world may be constructed by first sensing the position of the target part. Or, more or less sophisticated attempts may be used to cope with failures. For example, if a gross motion or grasp plan fails, one may attempt to move parts so as to make more room. These particular extensions have in fact been implemented within the HANDEY framework. Our emphasis in the remainder of the book, however, will be on the design and implementation of the four basic planners that form the core of HANDEY. Before we do that, we first review some of the relevant literature.

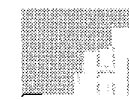
2.5 Previous work

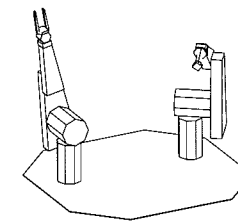


Task-level robot systems have long been a goal of robotics research. As early as 1961, Ernst's Ph.D. thesis at MIT [18, 74] attempted to develop such a system. During the 1970's, a variety of task-level robot systems were proposed and saw various levels of implementation. The Stanford Hand-Eye system [19] was very influential in the development of modern-day robot control systems and programming languages, notably through the work of Paul [62]. Taylor's work in the context of the AL system, also at Stanford, explored the problem of synthesizing sensor-based assembly strategies from models, one of the central problems in task-level robot systems. The LAMA system developed at MIT by Lozano-Pérez [43] proposed a design bridging path planning, grasp planning and synthesis of sensor-based strategies, but the system was only partially implemented, with the main focus on grasp planning. The AUTOPASS system at IBM [42, 77] was even more ambitious in scope, but implementation work focused only on the path-planning component [51]. The RAPT system of Ambler and Popplestone at Edinburgh [67, 68] provided the programmer with facilities for specifying robot positions by indicating symbolic relationships between geometric entities in a model of the task. RAPT, however, did not attempt to plan robot motions. The LM-GEO system of Mazer at LIFIA (Grenoble) [54] followed an approach similar to that of RAPT.

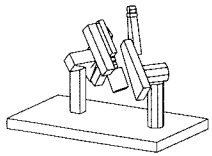


None of these projects produced a task-level robot system that could perform reasonably complex pick-and-place or assembly operations without substantial human programming. In retrospect, the problems were partly the result of lack of good algorithms for basic problems, such as motion planning and grasping, and partly the result of the inadequacy of available computers. Nevertheless, these systems helped define many of the key problems in task-level planning systems and motivated a great deal of subsequent work on algorithms during the 1980's.



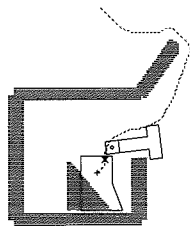


Only recently, after nearly a decade devoted to the development of algorithms for motion-planning, grasping and error propagation, and after a twenty-fold increase in computer speed, have there been new attempts at constructing complete task-level systems. The SHARP system [39] under development at LIFIA (Grenoble) incorporates a motion planner, a grasp planner and an assembly planner based on uncertainty modeling. The SPAR system [32] under development at Purdue is an assembly



planner that focuses on the simultaneous satisfaction of uncertainty reduction and geometric constraints expressed symbolically. The system of Violero et al. [76] under development at LAAS (Toulouse) is the closest to HANDEY. They report a system that satisfies most of the constraints on the pick-and-place operation. They select grasps on the basis of local object geometry, nearby objects at the pickup point, the kinematic feasibility, and the relative gripper/object uncertainty. Furthermore, their grasp planner plans a kinematically feasible approach motion and a departure motion. Here departure means the gripper carries the object away from the pickup point. They use a configuration space method for computing the motion of the gripper both approaching the part and moving away with it. Finally, they check the path computed for collisions between the robot arm and environmental objects.

For more detailed surveys of previous work in task-level systems, see [45, 50].

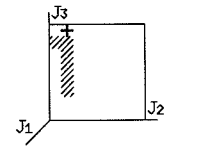


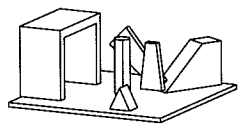
2.5.1 Motion planning

Work in developing task-level robot systems builds upon the very large body of work in motion planning. It is impossible to provide a comprehensive review of this work in a short monograph. Fortunately, there are several excellent reviews of work in motion planning. The most complete review of the motion planning literature is the book *Motion Planning* by J. C. Latombe [36]. There are several other shorter and, therefore, less comprehensive reviews, such as [55, 72, 78, 85]. Some representative gross-motion planning papers that are directly relevant to the approach taken in HANDEY are [5, 20, 47].

The vast majority of work in the motion planning literature addresses the planning of single types of motions, such as gross motions, fine motions, and stable grasping. The key problem in constructing a practical task-level system is considering the interactions between the different types of motions. The approach taken by HANDEY and all other proposed task-level robot systems is based on decomposing the planning of robot operations into nearly independent phases, e.g. gross-motion and grasping, and using different planners in each phase. This division gives rise to the interaction problems we have seen earlier in this chapter. There is work in the algorithmic motion planning literature that addresses complete pick-and-place problems in an integrated framework. This work has addressed a simple version of choosing a grasp as an

integrated part of planning a motion from an initial state to a final state in which the grasped object is placed at a destination [2, 40, 41, 80]. Existing implementations of these algorithms are for planar objects that can only translate. Nevertheless, they provide considerable insight into the computational complexity of the problem of planning pick-and-place motions.

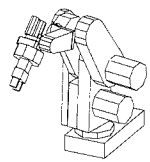




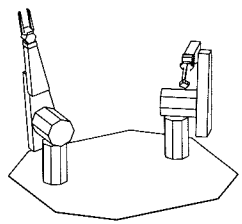
One crucial area of motion planning where HANDEY has not ventured is the area of producing plans that are robust to uncertainty. There has been a great deal of work in this area, most of which is reviewed in [36]. The interested reader is referred to [7, 12, 15, 16, 49, 53] for a sample of some of this work.

2.5.2 Grasp planning

A substantial body of research has been done on grasp planning. Pertin-Troccaz [65] has published a very comprehensive annotated bibliography of this literature. Most of this work has focused on grasping a part in a particular environment without considering all the constraints required to ensure the success of a pick-and-place operation.

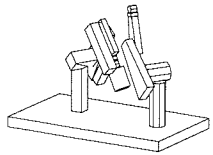


The work most relevant to HANDEY involves the planning of grasps for parallel-jaw grippers. Some representative early work on grasp planning for parallel-jaw grippers is: [8, 37, 38, 43, 44, 62, 73, 82]. Most of this work focused on geometric interactions, although some consideration has been given to kinematic constraints [62, 73]. Wolter, et al., [83] considered stability as well as local reachability in choosing grasps. Pertin-Troccaz [64] considers on-line planning based on sensory data. Sets of feasible grasps (based only on the geometry of the object) are computed, then a laser-camera system is used to find local obstacles near the plane of the grasp (defined by the grasp features). A configuration space method is employed to plan the motion of the gripper to the grasp point. Finally, as we mentioned earlier, Violero et al., [76] report on a method that satisfies most of the constraints required for successful pick-and-place operations.



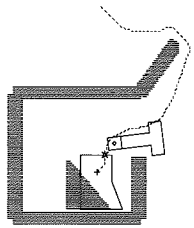
One crucial problem in grasp planning that HANDEY neglects is the problem of grasp stability, specially in the context of multi-finger grasping. There is a large body of published work on this problem, some examples are: [1, 3, 4, 11, 21, 27, 29, 33, 57, 58, 59].

2.5.3 Regrasp planning



If the robot is equipped with a multi-finger hand, it may be possible to change a grasp entirely within the hand [6, 21, 61]. If the robot is equipped only with a parallel-jaw gripper, however, an alternative strategy must be used that temporarily places the part on the table or in another gripper. The HANDEY regrasp planner is limited to this second strategy. As far as we are aware, however, the only previous systematic exploration of regrasping with parallel-jaw grippers was by Paul [62] in the context of the Stanford Hand-Eye system. Paul's method assumed that the regrasping could be done in a single step and did not deal completely with the kinematic or collision-avoidance constraints. Many of the ideas in HANDEY's approach to regrasping, however, can be traced to Paul's work.

2.5.4 Multi-robot coordination

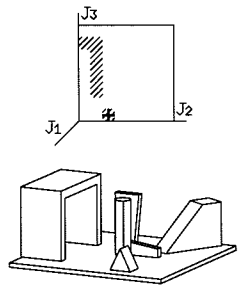


HANDEY can coordinate multiple robots operating in a common workspace. Existing approaches to multi-robot coordination can be classified as either *global* or *local*. The global methods construct complete trajectories for all the robots that guarantee, or attempt to guarantee, that all the robots reach their goals safely. (The multi-robot planner uses a global method.) Erdmann and Lozano-Peŕez [17] describe how to construct the configuration space-time for several planar manipulators, each with two revolute joints. The trajectories of the manipulators are planned one at a time, using the swept volume, in space-time, of the previous trajectories as obstacles. Fortune, Wilfong, and Yap [22] describe an algorithm for finding collision-free trajectories for two planar manipulators, with one prismatic and one revolute joint, by characterizing the combinatorial structure of the configuration space of the two robots.

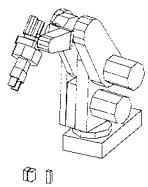
One problem with these global methods is that they depend on carefully controlled trajectories. There are many applications where this lock-step coordination is not practical or efficient (such as where the robots are controlled by hardware that cannot be tightly synchronized). Also, the methods are computationally intensive since they have to consider, either explicitly or implicitly, both space and time.

There has also been work [30, 70, 71] on the closely related problem of planning trajectories for multiple moving objects, not manipulators.

These studies have shown that this problem is PSPACE-hard, that is, one expects the time to achieve coordination to grow exponentially with the number of moving objects. This is not surprising; it has been shown repeatedly that the complexity of motion planning grows exponentially with the number of degrees of freedom of the task. There has also been work in the less directly relevant area of planning the motions of a single object in the presence of other moving objects whose trajectories are known [34].

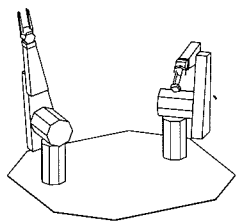


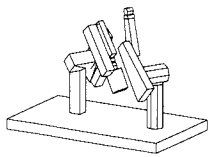
Local methods for collision-avoidance and coordination make decisions at each instant of time as to what the trajectory for each robot should be. These methods are related to local methods for avoiding collisions with fixed obstacles, notably the potential field methods originally developed in [35]. Freund and Hoyer [23] develop a controller that coordinates multiple planar manipulators, with one prismatic and one revolute joint, by incorporating collision constraints into the control system of the robots. Tournassoud [75] describes a technique for coordinating multiple moving objects, including manipulators, by defining separating planes at each moment and ensuring that the objects stay on opposite sides of them.

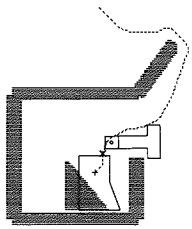


These local algorithms are based on actual measurements of the positions of the robots and thus can accomodate unexpected variations in trajectories or unexpected obstacles. But, in the general case, local methods cannot guarantee that each robot will reach its goal. They may reach a **deadlock**, where one robot is blocking the other. Furthermore, because these local methods rely on changing paths to avoid collisions they are not suited to situations where the paths are tightly constrained.

All the methods described above plan the paths of the robots as well as the trajectories. (Section 7.1.2 discusses the distinction between a path and a trajectory.) The multi-robot planner only schedules paths that have been planned by a different planner. In fact, in HANDEY, the paths that the multi-robot planner is given to synchronize are generated by at least two other HANDEY planners (the path planner and the grasp planner).

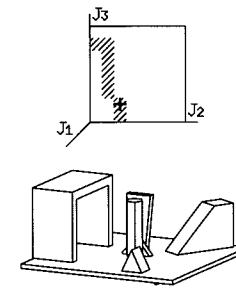






3 Basics

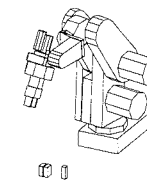
The basic problem that HANDEY faces is that of planning the motions of **robots** in the presence of obstacles—usually rigid objects, but sometimes other **kinematic chains**. HANDEY solves this problem by maintaining an explicit geometric model of its environment, computing an explicit representation of the constraints on the motion of the robot, and then searching for a path that satisfies these constraints. A variety of specialized representations are needed at various stages in this process, such as grasping and regrasping, and they will be discussed in subsequent chapters. There are a few basic representations, however, that permeate all the computations, and these are the subject of this chapter.



The representations that we will discuss in this chapter are:

- the representation of polyhedral models,
- the representation of robot models,
- the representation of world models, and
- the representation of motion constraints in configuration space.

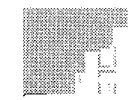
Although each of these could be (and each has been) the subject of its own book, we will only provide a few basic concepts necessary for understanding the operation of HANDEY.

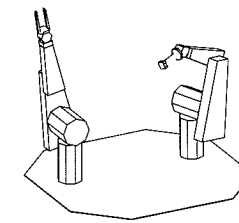


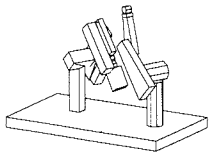
3.1 Polyhedral models

HANDEY operates in the domain of polyhedral solids. In this section, we briefly introduce the representation of polyhedra.

A **polyhedron** is a solid bounded by planar faces. In particular, a convex polyhedron is simply the common intersection of a set of **half-spaces** bounded by oriented planes. Each such plane is characterized by an **outward-pointing normal** vector and an offset that describes the perpendicular distance from the coordinate system origin to the plane. If the outward normal is the unit vector $\hat{\mathbf{n}}$ and the perpendicular distance to the origin is $-d$, then $\hat{\mathbf{n}} \cdot \mathbf{x} + d$ is the (signed) perpendicular distance of the point $\mathbf{x}$ from the plane. Points with positive distance are "outside" of the half-space defined by the plane and points with negative distance are "inside" the half space. Points with distance equal to zero are on the plane. Points in the interior of a convex polyhedron are inside all the half-spaces of the bounding planes.







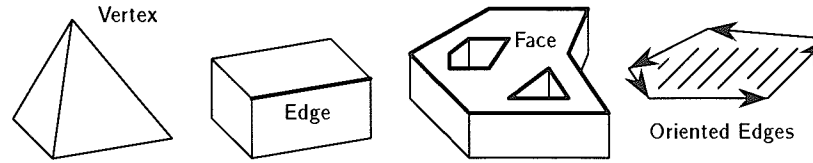
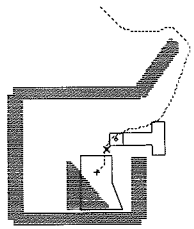


Figure 3.1
The representation of a polyhedral model.



Although the representation of convex polyhedra as a set of bounding planes is convenient for many purposes, such as detecting whether a point is inside or outside a body, it is not convenient for many other purposes, notably for graphics. For these other purposes a representation that explicitely characterizes the boundary of the polyhedron is required. Such a **boundary representation** is typically structured as a graph showing membership and adjacency relationships between the vertices, edges, and faces of the polyhedron. One notable advantage of this type of representation is that it readily extends to non-convex polyhedra.

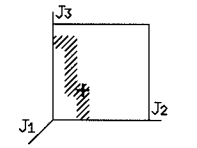
The fundamental elements in the boundary representation of a polyhedral model are:

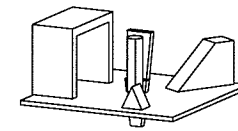
- **Vertices** are the points where three or more faces of the polyhedron meet. Vertices are represented by their x, y, z coordinates.
- **Edges** are the line segments where two faces meet. The edge representation stores pointers to the pair of vertices at the endpoints of the line segment and pointers to the two faces adjacent to the edge.
- **Faces** are planar polygons, bounded by the edges of the polyhedron. A polyhedral face is represented by the equation of the plane that contains the face, that is, by $\hat{\mathbf{n}}$ and d, and by a set of nested loops of oriented edges; the nesting and consistent orientation of the edges are necessary to enable the representation of objects with holes.

The representations of vertices and edges are intuitive; faces are more difficult and, therefore, we will describe them in more detail.

A simple polygonal face is represented by a loop of **oriented edges**. The edges are oriented so that as we traverse the edge from its tail to its head, the interior of the face is on the left (see Figure 3.1). For the outermost loop of a face, this orientation defines a counterclockwise

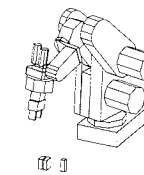
traversal of the face boundary (as seen looking from the exterior of the polyhedron). Note that since an edge is shared by two faces, it appears in two different face loops but in opposite orientations. Because the edge loops are consistently oriented, they can also be used to bound holes in the face. A hole in a face is bounded by another oriented loop. We still retain the condition that traversing the edges of the loop should keep the interior of the loop on the left. The result is that a hole boundary defines a clockwise traversal of the loop.





A polyhedral model, therefore, is composed of a set of faces, edges, and vertices as described above. Each face is described as a set of loops depicting the polygonal shape of the face. Each loop is described as a list of edges (and orientation) each of which is described as a pair vertices. In addition, there are several cross-references within the model: vertices point to the edges and faces incident on them and edges point to the faces that meet there. The resulting graph is the polyhedral model.

3.2 Robot models

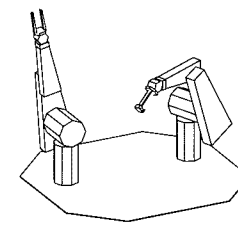


The description of a robot has a number of detailed components but they fall into two principal categories: kinematic and geometric. The kinematic description focuses on the motions of the robot and the geometric description focuses on the shape of the robot.

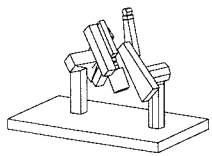
3.2.1 Kinematics

Robot manipulators are composed of a sequence of solid **links** connected by **joints**. The joints are of two principal types:

- **Revolute joints** permit the relative rotation of two adjacent links about a common axis. The relative orientation of the links is characterized by a **joint angle**.
- **Prismatic joints** permit the relative displacement of two adjacent links along a fixed axis. The relative displacement between the links is characterized by a **joint displacement**.

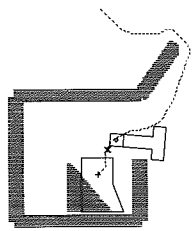


In this book, we focus on manipulators composed of revolute joints and we use the term "joint angle" to refer to the parameters that describe the position of a robot. The extension to prismatic joints is, in most cases, straightforward.



The kinematic description of the robot includes a description of the spatial relationships between the links of the robot, a means for selecting the joint angles of the robot given a desired end-effector location, and the specification of mechanical limits on the motion of the joints.

For the purpose of describing the motions of the robot, we do not need to know the shape of the links; all that is required is to know the relative positions and orientations of the joint axes. This is a **kinematic** description of the robot. The details of the kinematic description of robots is beyond the scope of this book; the interested reader may refer to [10, 63]. For our purposes, it is sufficient to understand that the robot kinematics can be described by a sequence of **coordinate transformations**. These transformations are typically represented by homogeneous coordinate transformation matrices [63], traditionally called **A matrices**. Each link has attached to it a coordinate frame whose origin lies on the joint axis closer to the base of the robot. The z axis of the link coordinate frame is aligned with the joint axis; therefore, the rotation of the joint amounts to a rotation about the z axis of the link frame. When all the joint angles are zero, the link frames have fixed relationships to each other. The i^{th} **A** matrix, $\mathbf{A}_i$, describes how to map the coordinate representation of points expressed in the frame of link i to the corresponding coordinate representation in the frame of link $i-1$. By composing subsequences of these transformations one can map points between any pair of link frames, including the frame of the fixed base of the robot.

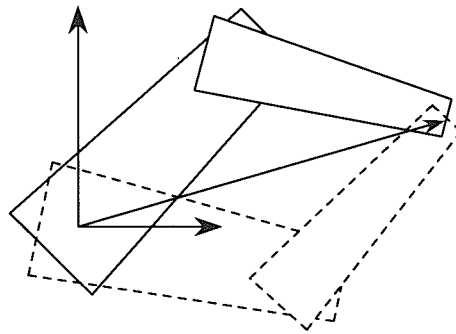


When the robot is positioned by setting the joint angles, the relationships between adjacent links is no longer defined by the constant **A** matrices described above. We must take into account the rotation about the joint axes. This is done by composing the constant A matrices we have discussed with a rotation about the link frame's z axis to obtain a new **A** matrix that depends on the joint angle:

$$\mathbf{A}_i(\theta_i) = \mathbf{A}_i\mathbf{Rot}(\widehat{\mathbf{z}}, \theta_i)$$

This matrix maps points from the link i frame to the link $i-1$ frame for any value of the joint angle. Composing these matrices for particular values of the joint angles enables us to position the robot model as desired.

A key aspect of the robot kinematics that remains to be addressed is the problem of choosing the set of joint angles that will place the



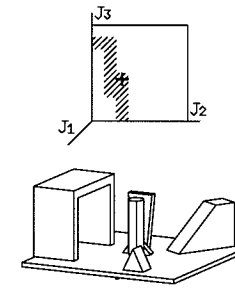
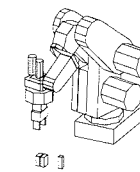
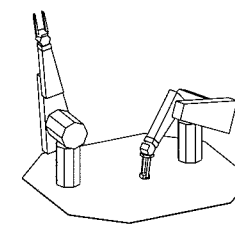


Figure 3.2
A two-link revolute manipulator can reach typical points in its workspace with two solutions. A general six-degree-of-freedom robot can have up to 16 distinct solutions to its inverse kinematics.

robot end effector at a desired pose, specified by a coordinate frame relative to the base of the robot. This is known as the **inverse kinematics** problem. There is no closed-form solution to this problem for arbitrary robots [10, 63]. We assume that a procedure is available to solve this problem; this procedure is part of the robot's kinematic description. It is crucial to note that there are, in general, *multiple solutions* for any gripper pose. For example, the planar two-link revolute manipulator in Figure 3.2 typically has two solutions for each gripper frame in its workspace. A six-degree-of-freedom Unimation Puma can reach a specified gripper frame in 8 different configurations. A general six-degree-of-freedom robot can have up to 16 distinct solutions to its inverse kinematics. Which solution should be chosen will depend on the environment.



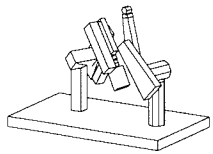
Another crucial component of the kinematic description of a robot is its **joint angle limits**. This is simply a description of the ranges of joint angles that are legal. These joint limits have a tremendous impact on what operations are possible and must be taken into consideration during all the planning operations.



3.2.2 Link shapes

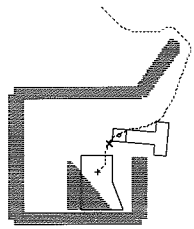
The description of the motion of the robot and the position of the end-effector depends only on the kinematic description of the robot. The

shape of the links of the robot is immaterial. Nevertheless, when attempting to plan a path for the robot that avoids obstacles in the world, the shape of the links becomes *very* important. Therefore, along with the kinematic description, the robot model includes a geometric description of the shape of the robot.



In the geometric description of the robot, each link is represented by a polyhedral model (Section 3.1). The model describes the shape of the link. The model is defined in the coordinate frame of the link, as defined in the previous section. For link i, we can then compute the link model in the coordinate frame of link $i-1$ simply by applying the $\mathbf{A}$ matrix, $\mathbf{A}_i(\theta_i)$. By this method, we can compute a model of the robot in the coordinate frame of the robot base for any specified joint vector.

3.3 World models

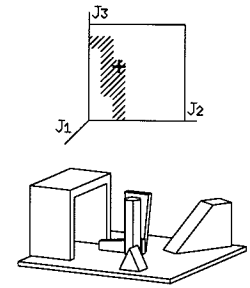


In order to plan the motions of a robot amid obstacles, HANDEY requires an explicit model of the robot's workspace (or "world"). This includes a robot model for each robot in the world, and a polyhedral model for every obstacle in the world. The parts being moved by the robots also need to be modeled. The collection of these robot and polyhedral models is a **world model**.

Specifying a world model to HANDEY merely requires the listing of a part model for each obstacle in the world. For each model, it is also necessary to specify the location in the world of that part. This is done by providing a coordinate transform that specifies the position and orientation of the coordinate system used to define the polyhedral model with respect to the world coordinate system. In other words, when the polyhedral model is defined, a reference coordinate system is assumed by the definition of the model; when the world model is defined, the relationship of this reference coordinate system to the world coordinate system is specified.

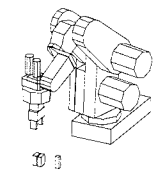
In HANDEY, the location and orientation of the robot is encoded within the robot model, specifically, within the description of the link numbered zero (the base). The actual location of any of the links of the robot in the world can be computed from the robot model, using the specific values of the joint angles, according to the method described in the previous section.

It is not necessary to define a detailed polyhedral model for every object in the robot's workspace. The detail that is necessary depends on the use to which the object is put. For example, the world may contain a group of small parts that are not expected to be manipulated and the robot is not expected to navigate among them—it is only expected to avoid the whole collection. In that case, it is convenient to represent the whole collection as a single obstacle, using a single polyhedral model whose interior contains the whole group.



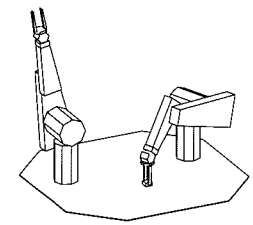
At the other end of the spectrum, any object that the robot is expected to grasp, to mate to other objects, or to move very close to, must be modeled very accurately. The grasp planner, for example, must have an accurate list of the planar faces of the object in order to decide on possible grasps.

In the middle of the spectrum are objects near to which the robot is expected to move, but that are not expected to be manipulated. These objects may be modeled with as much accuracy as is reasonable in order to allow motion around them, but details of their shape may be ignored. (Of course, the polyhedron approximating the obstacle must be conservative—the actual object must be contained within the interior of the approximation.)

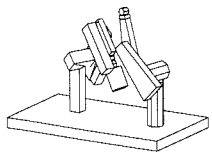


The accuracy of the models of the objects in the world directly affects the performance of many of the modules of HANDEY. The running time of these modules depends on the number of faces and edges in the entire world model. The more detail included in the world model, the longer these modules will run. Nearly all of them take steps to reduce the impact of detailed models, through caching of previous results or conversion of the information in the world model into other formats.

3.4 Configuration space

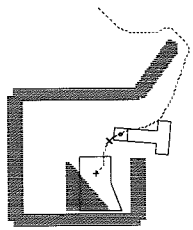


Many of the planning modules in HANDEY operate by computing the constraints on the motion of the robot due to the presence of obstacles in the world. These constraints are represented in a **configuration space map** (**C-space map**). The C-space map specifies which combinations of joint values (or x, y, θ) cause collisions with obstacles and which do not. The gross motion planner, for example, searches for a piecewise-straight path through the robot's C-space map that avoids all collisions.



If the robot were a point, then the obstacles in the C-space map would simply be the obstacles in the world but, for an actual robot, the relationship between the obstacles in the C-space map and the obstacle descriptions in the world model is complex. C-space maps play a prominent role in HANDEY. Therefore, in this chapter, we will present a detailed exposition of the concept of **configuration space** (**C-space**). We illustrate the concept by outlining algorithms for circles, polygons and polyhedra. The detailed algorithms used in HANDEY will be described in subsequent chapters.

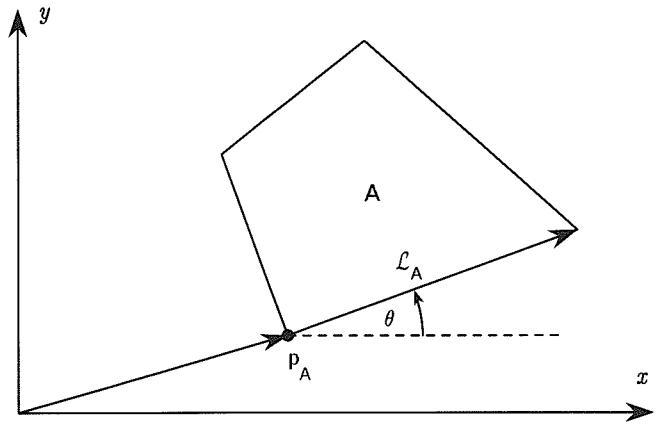
3.4.1 Definition of configuration space



The position of all points on a rigid body or robot manipulator can be specified by a small set of numbers. This collection of numbers is the **configuration** of the body. (For rigid bodies, this is also known as the **pose** of the body.) For example, the x, y position of a known point on a rigid planar body and an angle θ indicating the rotation of a line on the body about a fixed orientation are sufficient to specify the position of every point on the body. In this case, we can write the configuration as (x, y, θ). Of course, we can parameterize configurations in different ways, for example, using polar coordinates r, ϕ instead of x, y. Each choice of parameters for the configurations of an object defines a space, called the **configuration space** (C-space) of the object, where a point corresponds to a configuration of the object.

For jointed manipulators, the natural choice of configuration parameters is the set of joint parameters. Another obvious choice is the position and orientation of the gripper frame, but this choice is not always suitable since it does not uniquely identify the positions of the links for general manipulators. As we saw, the planar two-link revolute manipulator in Figure 3.2 (page 39) typically has two solutions for each gripper frame in its workspace.

If the moving object is a polygon A, there is no unique choice of configuration parameters, so we must adopt some arbitrary convention for representing its configuration. Such a convention amounts to choosing a coordinate frame attached to the moving object, that is, we must choose a **reference point**, $\mathbf{p}_A$, to determine position and a **reference line**, $\mathcal{L}_A$, to determine orientation (see Figure 3.3). The choice of point and line is arbitrary; in particular, they need not be inside of A. It is only necessary that the reference point and line bear a fixed relationship to A.



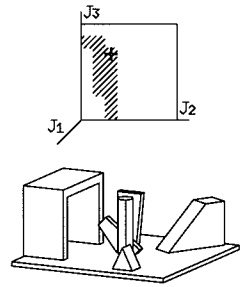
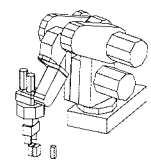


Figure 3.3
Conventions for assigning configurations to a polygon

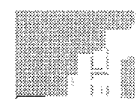
A's configuration, $\mathbf{c}$, is (x, y, θ) where x, y are the components of $\mathbf{p}_A$ and θ is the angle that $\mathcal{L}_A$ makes with the world x-axis. The zero position of a polygon has $\mathbf{p}_A = \mathbf{0}$ and the reference line parallel to the x-axis.

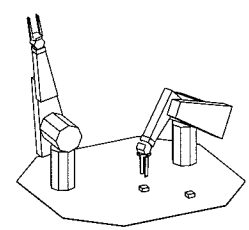


In the C-space of a moving object A, there will be some configurations that would place some point of A inside some Cartesian obstacle B_j. We call the collection of these configurations, the **configuration space obstacle** due to B_j, written as $CO_A(B_j)$. The definition is:

$$CO_A(B_j) = \{\mathbf{c} | (A)_\mathbf{c} \cap B_j \neq \emptyset\} \tag{3.4.1}$$

where $(A)_\mathbf{c}$ denotes A in configuration $\mathbf{c}$. We can think of $CO_A(B_j)$ obstacle as a new object obtained by mapping B_j into the C-space. Clearly, the space available for motions is that part of the C-space outside all the configuration space obstacles; we call this the **free space**.

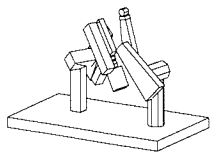


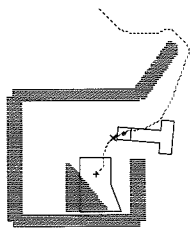


HANDEY must compute the C-space obstacles of realistic objects, including manipulators. But, we will first illustrate these concepts with very simple objects, such as circles and convex polygons.

3.4.2 The CO for a circle

If A is a point in the plane, the C-space is simply the plane itself and the configuration space obstacles (CO's) are the Cartesian obstacles themselves.





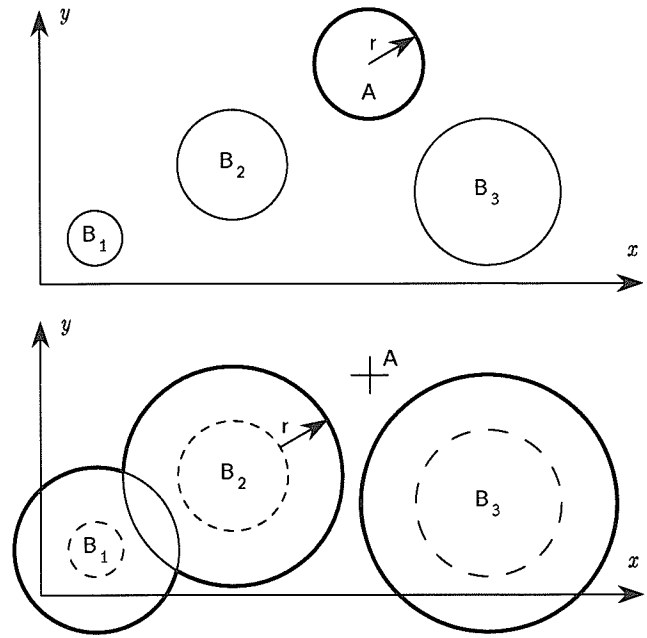
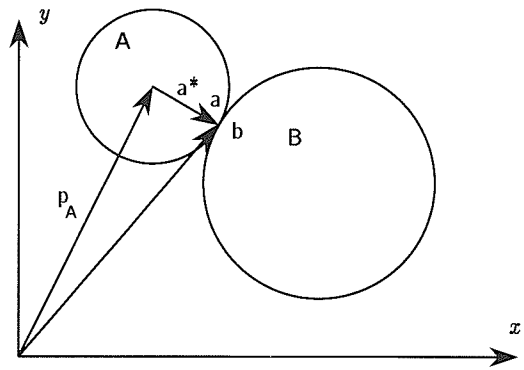


Figure 3.4
The configuration space obstacles (CO's) for a circle A amid circular obstacles B_j are the input obstacles grown by the radius of A.

If A is a circle, we can choose to represent its configuration by the position of its center point. Given this choice, then the circle's orientation is irrelevant and the C-space is the Cartesian plane. But, the CO's for the circle are not simply the Cartesian obstacles; any position of the circle that is within the circle radius from a Cartesian obstacle must be within a C-space obstacle. Therefore, the COs for the circle are the Cartesian obstacles *grown* by the circle radius.

If the obstacles are circles, the CO's are also circles whose radius has been increased by r_A, the radius of A; dually, the circle A shrinks to its center point in the C-space. (See Figure 3.4 for an example.) The circle is safely outside the Cartesian obstacles if and only if the position of its center point, that is, its configuration, is outside all the COs.

Let us look at the COs for a circle amid circular obstacles more carefully. Imagine tracing the path of A's center, $\mathbf{p}_A$, as A travels around a



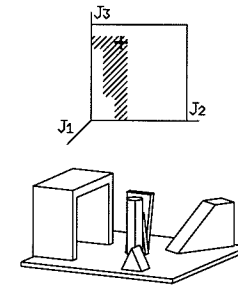
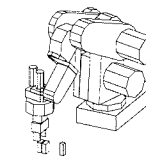


Figure 3.5
Computing the CO as a difference of vectors from the center to points with equal and opposite normals.

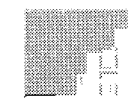
circular obstacle B while keeping contact with its boundary. The point of contact between the two circles occurs at point **a** on the boundary of A and point **b** on the boundary of B. A key observation is that the outward pointing normals at **a** and **b** must be equal and opposite for the points to touch (see Figure 3.5).

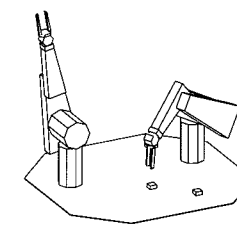


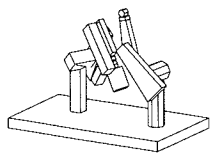
The positions of $\mathbf{p}_A$ can be characterized in another useful way. Let $\mathbf{a}^* = \mathbf{a} - \mathbf{p}_A$, that is, the position of contact point **a** relative to the reference point of A. Then, we know that if **a** and **b** are in contact (that is, $\mathbf{a} = \mathbf{b}$), then

$$\mathbf{p}_A = \mathbf{b} - \mathbf{a}^* \tag{3.4.2}$$

This gives the position of A in terms of the positions of the contact points. (3.4.2) holds for any pair of points on the boundary of A and B, $\mathbf{a}^*$ and **b**, that have opposing outward normals (see Figure 3.5). This relationship can be used to compute the CO boundary. Note that for any convex object, the angle defined by the outward normal (relative to the x-axis) increases monotonically as we navigate counterclockwise around the boundary. So, we could compute the CO boundary—the locus of positions of $\mathbf{p}_A$ for which A and B touch, in a single traversal of the boundaries of the objects, by using (3.4.2) where the normals are opposite. For circles the result is obvious, a grown circle, but this is a general characterization of the CO boundary for arbitrary objects.







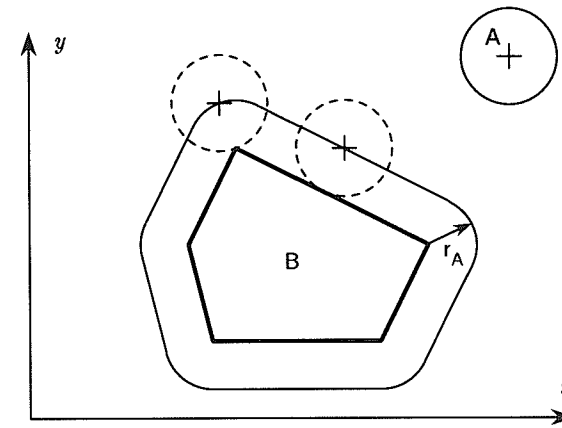
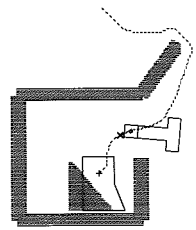
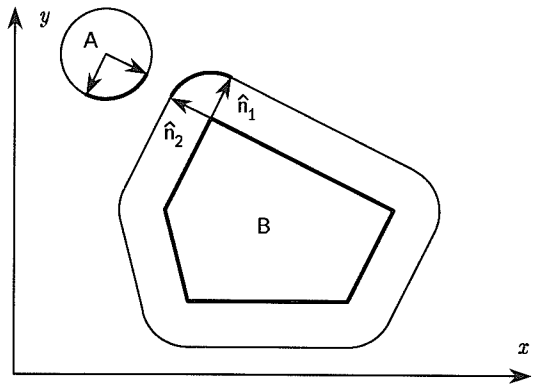


Figure 3.6
CO for a moving circle A and a polygonal obstacle B.



Consider the CO's for the circle A and a polygonal obstacle B, as shown in Figure 3.6. Examining the figure one can see that the boundary of the CO is composed from edges parallel to the edges of B and from arcs of A. Visualize the path of A's center as its boundary slides along the boundary of B while maintaining contact. Part of the time, the circle is in contact with an edge of the polygon, part of the time it is in contact with a vertex of the polygon. When touching an edge, the circle's center point traces an edge parallel to the polygon's edge but displaced by radius r_A. When touching a vertex, the circle's center point traces an arc of a circle of radius r_A centered on the vertex. This satisfies our intuitive expectations of what should be the boundary of the set of positions of $\mathbf{p}_A$ that could collide with B.

These conclusions about the shape of the boundary also follow from considering the normals at the points of contact. First, consider the edges of B. All the points $\mathbf{b}$ on an edge of B have a common normal, but only one point on A has that particular normal; hence the point of contact $\mathbf{a}^*$ is uniquely defined and remains constant as A translates along the edge. All the values of $\mathbf{p}_A$ obtained from (3.4.2) are on the CO boundary. These values define an edge parallel to B's edge, offset from it by the vector $\mathbf{a}^*$ which, because A is a circle, is perpendicular to the edge of B and of length r_A.



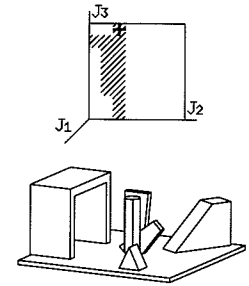
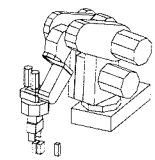
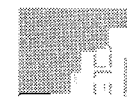
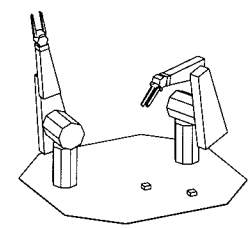


Figure 3.7
The boundary of the CO corresponding to positions of A for which A's boundary is in contact with a vertex of B is a displaced and negated copy of an arc from A's boundary. The range of normals of the arc is determined by the range of normals between the edges neighboring the contact vertex

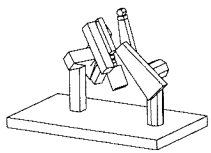


Now, consider the vertices of B. A vertex can be seen as the limiting case of a circle as the radius goes to zero. In this view, a vertex has the whole circular range of normals, but contact is only possible in the range between the normals of the edges connected to it (see Figure 3.7). The vertex has a constant position $\mathbf{b}$; hence, the path of $\mathbf{p}_A$ while in contact with the vertex is obtained by subtracting from $\mathbf{b}$ the range of vectors $\mathbf{a}^*$ for points on A with opposing normals. This is simply a displaced (and negated) copy of an arc from A's boundary, as illustrated in Figure 3.7. Because of A's symmetry, the negative sign in (3.4.2) has no discernible effect but, in general, it is quite important, as we will see below.





Note that the circle and convex polygon (using the convention about vertex normals used above), have a normal corresponding to each angle in the interval $[0, 2\pi)$. Hence, we can place the objects in contact so that the contact point has any normal from this range. Also, for each angle, there is a single point on the boundary of each obstacle whose normal makes that angle with the world x-axis. By the reasoning in the preceding paragraph, we can expect that an edge equal in length and parallel to each edge on B's boundary will appear on the boundary of the CO. Similarly, the union of the arcs generated at the vertices will yield a full circle.



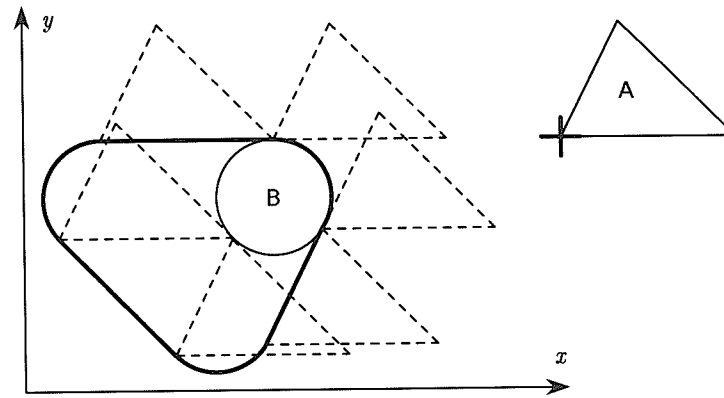
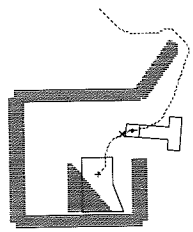


Figure 3.8
CO for a moving polygon A and a circular obstacle B.

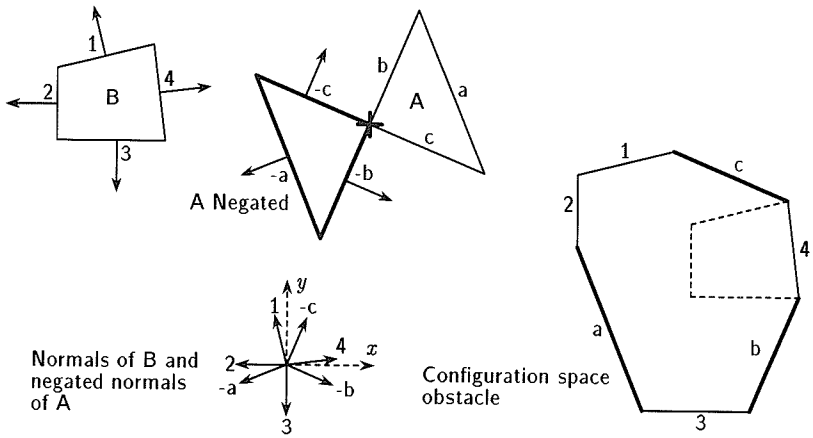


3.4.3 The translational CO for convex polygons

A polygon has three degrees of motion freedom: two translations and one rotation. Thus, the general CO for a moving polygon A is a three-dimensional entity. We will consider only a single orientation $\theta = 0$ of A, that is, a cross section of the general CO. In general, we denote such cross sections $CO^{\theta=0}$, but we will not use the superscript when the reference is clear from the context.

CO's for a single convex moving polygon For a polygon A and a circular obstacle B, we expect that the translational CO will be quite similar to the case of a moving circle B and a polygonal obstacle A. This expectation is not completely correct. It is indeed the case that the CO boundary for a polygon A and a circle B is composed of arcs of B and edges equal in length and parallel to those of A. But, the normals to the edges of the CO will be opposite to those of the corresponding edges of A (see Figure 3.8). The case of a circular A demonstrated the same effect, but it was not noticeable because of symmetry. The reversal follows from the negative sign on $\mathbf{a}^*$ in (3.4.2).

The resulting $CO_A^{\theta=0}(B)$ is rotated by π radians from $CO_B^{\theta=0}(A)$. Let $\mathbf{p}_B$ be a reference point for B and $\mathbf{b}^*$ the position vector of a boundary point on B relative to the reference point. Essentially, every



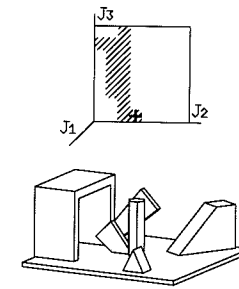
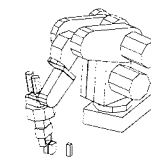


Figure 3.9
CO for a moving polygon A and a polygonal obstacle B. The edge normals of B and the negated edge normals of A can be sorted by angle; this order will be the order of their appearance in the *CO* boundary. The range between two edge normals is associated with a vertex.



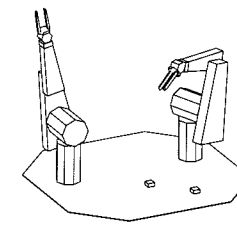
boundary point in $CO_A^{\theta=0}(B)$ is expressible as

$$\mathbf{p}_B + \mathbf{b}^* - \mathbf{a}^* \tag{3.4.3}$$

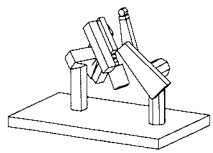
while every boundary point in $CO_B^{\theta=0}(A)$ is

$$\mathbf{p}_A + \mathbf{a}^* - \mathbf{b}^* \tag{3.4.4}$$

Assume that both A and B are in their initial position so that $\mathbf{p}_A = \mathbf{p}_B = \mathbf{0}$. Then, the boundary points of one *CO* are the negatives of the other, that is, one *CO* is the reflection through the origin of the other. In two dimensions, this reflection is equivalent to a rotation of π radians. When the positions of A and B are changed, the shape of the *CO* is unchanged; the only effect is a translation of the *CO*.

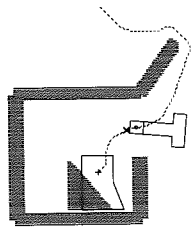


When both A and B are convex polygons, the translational *CO* is also a convex polygon. From the above discussion it should come as no surprise that the boundary of this *CO* is composed of edges equal in length and parallel to the edges of A and B. For an example, see Figure 3.9. Clearly, unless an edge of A and one of B have opposing normals, contact between A and B will involve an edge of one touching



a vertex of the other. We classify the CO edges by which object provides the edge. When an edge of A touches a vertex of B, we call the resulting CO edge a **type A edge**. When an edge of B touches a vertex of A, we call the resulting CO edge a **type B edge**. Type A edges have opposite normals from the corresponding original edges of A. When A and B have edges with opposing normals, the resulting CO edge is the union of a type A and type B edge.

Each CO edge is obtained from a pairing of an edge from one object and a vertex of the other. The particular pair that actually generates such an edge is determined by the normal of the edge and the range of normals of the vertex (as defined earlier). Only when the range of normals associated with a vertex includes the negated normal of an edge can that vertex–edge pair come into contact and produce a CO edge (see Figure 3.9).



The condition for a vertex–edge pair forming a CO edge leads to a very simple algorithm for the translational CO for polygons. The vertices that define a polygon are stored in order of increasing angle; this provides a convenient data structure for implementing the algorithm. Define a loop that, in parallel, scans the vertex list of both polygons. Conceptually, an angle cursor is advanced each time through the loop, succesive values of this cursor correspond to the angle of the edge normals of B and the negated edge normals of A. At any point, the angle cursor will point to a vertex of one object and an edge of the other.

Having obtained the vertex–edge pairings, the resulting CO edge is determined by an application of (3.4.2). Consider a type B edge first. Let the position vectors of endpoints of the edge of B be $\mathbf{b}_j$ and $\mathbf{b}_{j+1}$ respectively and let the position vector of the corresponding vertex of A (relative to the reference point of A) be $\mathbf{a}_i^*$. Then the endpoints of the edge of $CO_A(B)$ generated by this pairing are: $\mathbf{b}_j - \mathbf{a}_i^*$ and $\mathbf{b}_{j+1} - \mathbf{a}_i^*$.

We represent a line supporting an edge by an equation of the form: $\widehat{\mathbf{n}} \cdot \mathbf{x} + d = 0$. Where $\widehat{\mathbf{n}}$ is the (outward pointing) unit vector that is normal to the line and the absolute value of d is the perpendicular distance to the edge from the origin. If the line equation of the j^{th} edge of B is:

$$\widehat{\mathbf{n}}_j \cdot \mathbf{x} + d_j = 0$$

then the line equation of the corresponding edge of the CO is:

$$\widehat{\mathbf{n}}_j \cdot (\mathbf{x} + \mathbf{a}_i^*) + d_j = 0 \tag{3.4.5}$$

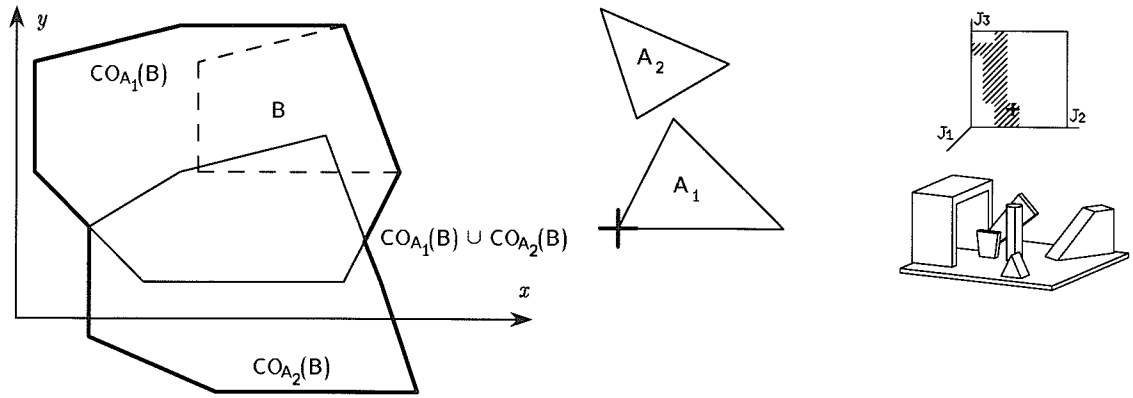
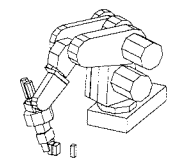


Figure 3.10
CO for $A = \cup A_i$ for obstacle B.

Note that this equation is satisfied by the endpoints of the CO edge, as it must.



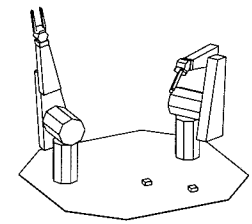
The situation for type A edges is analogous. The position vectors (relative to the reference point of A) of the endpoints of the i^{th} edge of A are $\mathbf{a}_i^*$ and $\mathbf{a}_{i+1}^*$; the position vector of the B vertex matched to that edge is $\mathbf{b}_j$. Then the endpoints of the edge of $CO_A(B)$ generated by this pairing are: $\mathbf{b}_j - \mathbf{a}_i^*$ and $\mathbf{b}_j - \mathbf{a}_{i+1}^*$. Similarly, if the line equation of the i^{th} edge of A is:

$$\hat{\mathbf{m}}_i \cdot \mathbf{x} + c_i = 0$$

then the line equation of the corresponding edge of the CO is:

$$\hat{\mathbf{m}}_i \cdot (\mathbf{b}_j - \mathbf{x}) + c_i = 0 \qquad (3.4.6)$$

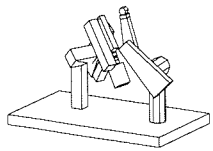
Note that once again this equation is satisfied by the endpoints of the CO edge.



CO's for unions of convex polygons The methods described above for a single moving polygon generalize directly for moving objects representable as unions of convex polygons. In that case, the CO is simply the union of the CO's of each of the component polygons. For an example, see Figure 3.10. It is crucial that a single reference coordinate system be used for all the polygons in the union. The effect is to refer

the motion constraints of all the components of the moving object to a single point.

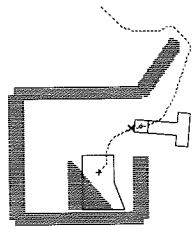
The CO algorithm suggested in Section 3.4.3 can be generalized to handle non-convex obstacles [26], but the resulting CO is a polygon whose boundary, in general, crosses itself. Such a self-intersecting polygon is called a **non-simple polygon**.



3.4.4 The CO for convex polyhedra

We now turn our attention briefly to the problem of computing $CO_A(B)$ when A and B are convex polyhedra. The translational case is a fairly direct extension of the polygonal case, but the general case including rotations is much more difficult.

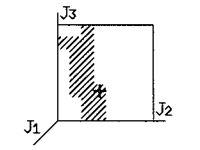
When both A and B are polyhedra, the faces of $CO_A^{\theta=0}(B)$ are of three basic types, the first two are analogous to the type A and type B edges we have already seen, the third is new.

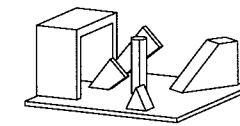


1. Type A faces — the set of positions of A for which a face of A is in contact with a vertex of B. The normal to the C-surface is the normal of the face of A that gives rise to it.
2. Type B faces — the set of positions of A for which a face of B is in contact with a vertex of A. The normal to the C-surface is the normal of the face of B that gives rise to it.
3. Type C faces — the set of positions of A for which an edge of A is in contact with an edge of B. The normal to the C-surface face is parallel or anti-parallel to the cross product of the edge vectors that give rise to it, that is, the C-surface normal is orthogonal to both edges.

When both A and B are convex polyhedra, we can readily determine which of the pairings of edges, faces, and vertices actually produce faces on the boundary of the CO. The conditions for type A and B surfaces are simply that the edges that intersect at the contact vertex all be above the plane of the contact face. If any of the edges at the point of contact were below the plane of the face then that would indicate that the two objects would be overlapping, not just in contact at that point. In general, the applicability condition, for convex A and B, requires that the C-surface act as a separating plane between the two solids, that is, that A and B be on opposite sides of the C-surface. The conditions for type C, although a bit more complex, follow this same pattern [12].

Three-dimensional solids have six degrees of motion freedom and, therefore, they require a six-dimensional C-space. The C-space obstacles are six-dimensional solids bounded by five-dimensional C-surfaces, of type A, B, and C. It is possible to compute descriptions of the bounding surfaces of these solids and to build motion planning algorithms on these descriptions [9, 12].

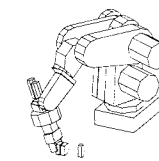




3.4.5 Slice projection

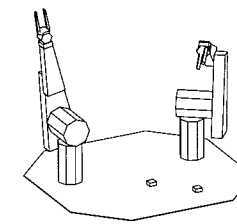
The C-space for a rotating planar object is three-dimensional, with one dimension corresponding to each of the degrees of motion freedom of the object. The C-space obstacles are also three-dimensional; they describe the range of forbidden configurations of the moving object. The translational CO's we have been discussing correspond to two-dimensional cross sections, for fixed orietations of A, of this three-dimensional CO.

It is possible to construct exact representations of these high-dimensional C-space obstacles [9, 12], but for many practical applications it is sufficient to build approximations to the exact C-space obstacle. One common approximation technique is to quantize one or more of the C-space parameters to obtain a lower-dimensional obstacle that represents the motion constraints over some range of values of that parameter. We call this technique **slice projection**. Slice projection is the basis of the gross motion planner in HANDEY. (See Section 4.1.)



Definition Let C denote an n-dimensional general C-space obstacle for a moving object with n degrees of freedom. We can represent approximations of C by the union of $(n-1)$-dimensional slice projections. Each $(n-1)$-dimensional configuration in a slice projection of C represents a range of n-dimensional configurations (differing only in the value of a single configuration parameter) that intersects C.

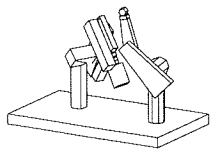
A slice projection of an n-dimensional C-space obstacle is defined by a range of values for one of the defining parameters of the C-space and an $(n-1)$-dimensional volume. Let $\mathbf{q} = (q_1, \ldots, q_n)$ denote a configuration, where each q_i is a configuration parameter, which measures either angular or linear displacement. Let π_j be a projection operator for points, defined such that



$$\pi_j(\mathbf{q}) = (q_1, \ldots, q_{j-1}, q_{j+1}, \ldots, q_n)$$

Let $\Pi_{q_j \in [a_j, b_j]}(S)$ be a projection operator for point sets S, defined such that

$$\Pi_{q_j \in [a_j, b_j]}(S) = \{\pi_j(\mathbf{q}) | \mathbf{q} \in S \text{ and } q_j \in [a_j, b_j]\}$$



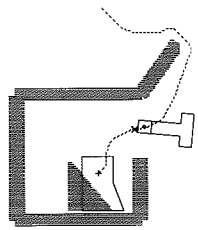
Then, the slice projection of the obstacle C for values of $q_j \in [a_j, b_j]$ is

$$\Pi_{q_j \in [a_j, b_j]}(C)$$

In the example above, configuration parameter j above is called the **slice parameter** while the other joints are known as **free parameters**.

Note that a slice projection is a *conservative* approximation of a segment of an n-dimensional C-space obstacle. That is, the slice projection contains all the points in the actual C-space obstacle and, usually, additional points not in the actual obstacle.

An approximation of the full obstacle can be built as the union of a number of $(n-1)$-dimensional slice projections, each for a different range of values of the same slice parameter. Each of the $(n-1)$-dimensional slice projections, in turn, can be approximated by the union of $(n-2)$-dimensional slice projections and so on, until we have a union of one-dimensional volumes (linear ranges) or possibly zero-dimensional ranges (points).

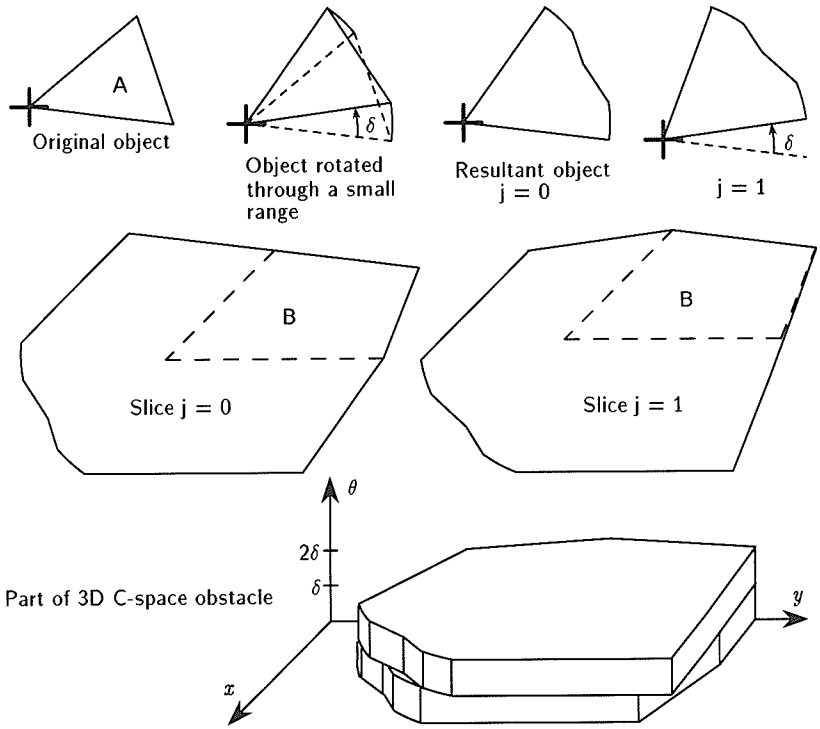


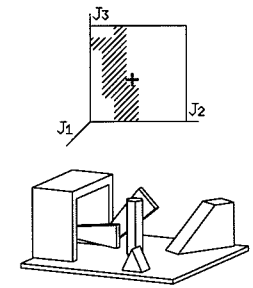
Computing slice projections A slice projection can be computed without having to compute the exact C-space obstacle first. The basic idea is very simple: A slice projection $\Pi_{q_j \in [a_j, b_j]}(CO_A(B))$ is merely the C-space obstacle for a suitably defined object $A_{q_j \in [a_j, b_j]}$ with one less degree of motion freedom than A. The object $A_{q_j \in [a_j, b_j]}$ is the volume swept out by A as q_j takes on all values in $[a_j, b_j]$. Figure 3.11 shows a slice projection for a range of orientations of A. Each slice projection is a two-dimensional polygon corresponding to the translational CO of $A_{\theta \in [0, \delta]}$ (for $j = 0$) and $A_{\theta \in [\delta, 2\delta]}$ (for $j = 1$).

Note that we can vary the size of the ranges used in the slice projection to control the accuracy of the representation. But, also note that some problems will require arbitrarily high resolution and therefore arbitrarily many slices.

3.4.6 The CO for manipulators

We have limited ourselves thus far to discussing Cartesian C-space obstacles, the obstacles for objects with Cartesian degrees of freedom. The





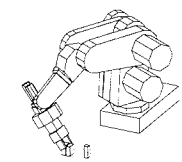
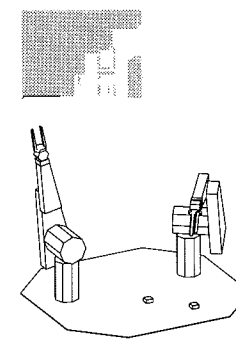


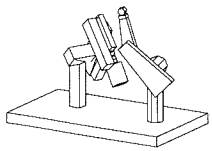
Figure 3.11
Slice projection of general $CO_A(B)$ for two ranges of θ: $\theta \in [0, \delta]$ (for $j = 0$) and $\theta \in [\delta, 2\delta]$ (for $j = 1$).

C-space concept is not limited to these cases; we have mentioned that the joint space of a manipulator is also a C-space. The C-spaces generated by manipulators with revolute joints, however, are more difficult to compute than those with prismatic joints, just as the rotational degrees of freedom lead to more complex CO's in the Cartesian C-spaces.

The natural choice of C-space for a manipulator is its joint space, since the Cartesian configuration of the end effector does not, in general, uniquely characterize the positions of the links.

The boundaries of the C-space obstacles for a manipulator can be obtained from the expressions for the type A and B edges in (3.4.5) and (3.4.6). Essentially, there are two types of collisions between links





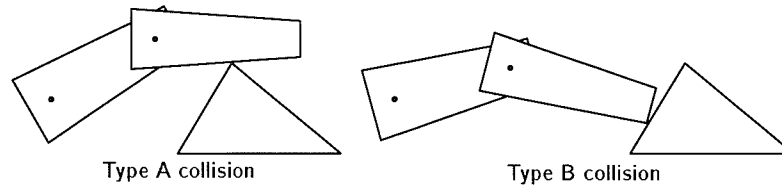
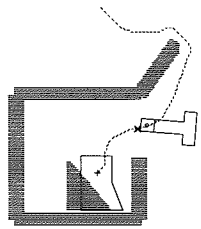


Figure 3.12
Two types of collisions: (type A) link edge and polygon vertex, (type B) link vertex and polygon edge.

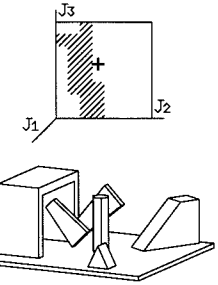
and polygonal obstacles: an edge of the link collides with a vertex of an obstacle polygon (type A) or a link vertex collides with an edge of an obstacle polygon (type B) (see Figure 3.12).



We can express the positions of the link vertices and edges as a function of the joint angles of the manipulator. Substituting these expressions in the equations of the CO edges in (3.4.5) and (3.4.6) give rise to implicit equations that are satisfied by joint angle vectors when there is contact between a link and obstacle. These equations describe the boundary surfaces (**C-surfaces**) of the C-space obstacles in the joint space of the manipulator [24]. Of course, these equations may be difficult to solve explicitly for the joint angles. Instead of pursuing this course, we will employ slice projection to construct numerical approximations of the C-space obstacles for manipulators (see Chapter 4).

4 Gross Motion Planning

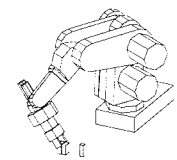
The HANDEY gross-motion planner plans collision-free motions for robots. The gross-motion planner is invoked as follows:



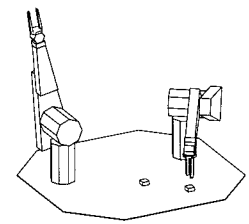
Move(*goal*, *robot*, *world*)

The planner computes a path for the robot from its configuration in the specified input world to the robot's configuration in the goal world. The path is required to avoid collisions with all the obstacles in the world model. The path produced by the planner is a list of robot joint vectors; the planner assumes that the robot controller can follow straight lines in the joint space between the specified configurations with good accuracy.

The planner operates by computing the constraints on the motion of the robot due to the presence of obstacles in the world. These constraints are represented in a **configuration space map** (**C-space map**) (see Section 3.4 for an introduction to configuration space). The C-space map specifies which combinations of joint values cause collisions with obstacles and which do not. The gross motion planner searches for a piecewise-straight path through the robot's C-space map that avoids all collisions.



This chapter first presents the slice-projection approach as applied to computing C-space maps for revolute robots. Next, we consider heuristic techniques for minimizing the amount of computation needed to compute C-space maps: reducing the number of joints in the maps, reducing the number of slices, simplifying the models of robots and obstacles, and caching intermediate results. Then we discuss how to find paths in the C-space maps. The chapter ends by presenting a gross-motion planning algorithm, based on the slice-projection approach, for implementation on massively parallel SIMD computers,[1] such as a Thinking Machines Corporation's Connection Machine.

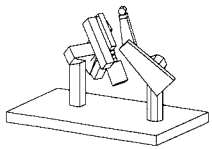


4.1 Approximating the *CO*s for revolute manipulators

Consider a simple two-link planar revolute manipulator whose joint parameters are θ_1 and θ_2. C-space obstacles for such a manipulator are two-dimensional. The one-dimensional slice projection of a C-space obstacle, C, for $\theta_1 \in [a, b]$ is some set of linear ranges $\{R_i\}$ for θ_2. The

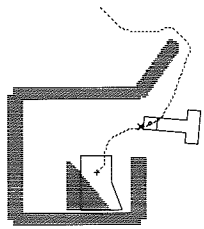
[1] Single Instruction Multiple Data—that is, a parallel computer whose many processors all receive the same instruction stream but operate on data local to each processor.

ranges must be such that if there exists a value of θ_2, call it d, and a value of $\theta_1 \in [a, b]$, call it c, for which $(c, d) \in C$, then d is in one of the R_i.



The configuration space of the two-link manipulator and obstacles as shown in Figure 4.1 (bottom), can be represented as in Figure 4.1 (top). The actual configuration space is the surface of a torus since the top and bottom edge of the diagram coincide ($0 = 2\pi$), as do the left and right edges. The obstacles are approximated as a set of θ_2 ranges (shown dark) for a set of values of θ_1. The resolution is two degrees along the θ_1-axis.

If the manipulator has three links, its configuration space can be constructed as follows:



1. Ignore links beyond link 1. Find the ranges of legal values of θ_1 by considering rotations of link 1 around the fixed base.
2. Sample the legal range of θ_1 at the specified resolution. Do steps 3 through 5 for each of the value ranges of θ_1.
3. Ignore links beyond link 2. Find the ranges of legal values of θ_2 by considering rotatations of link 2 around the positions of joint 2 determined by the current value range of θ_1.
4. Sample the legal range of θ_2 at the specified resolution. Do step 5 for each of these value ranges of θ_2.
5. Find the ranges of legal values of θ_3 by considering rotations of link 3 around the position of joint 3 determined by the current value ranges of θ_1 and θ_2.

The resulting C-space map (see Figure 4.2) is a set of two-dimensional slice projections, each like the one in Figure 4.1. Right-Margin Movie 1 shows steps in the construction of such a C-space map.

Note that the process described above is an instance of a simple recursive process:

To compute C-space(i):

1. Ignore links beyond link i. Find the ranges of legal values of θ_i by considering rotating link i around the positions of joint i determined by the current value ranges of $\theta_1, \ldots, \theta_{i-1}$.
2. If $i = n$ then stop, else sample the legal range of θ_i at the specified resolution. Compute C-space($i + 1$) for each of these value ranges of θ_i.

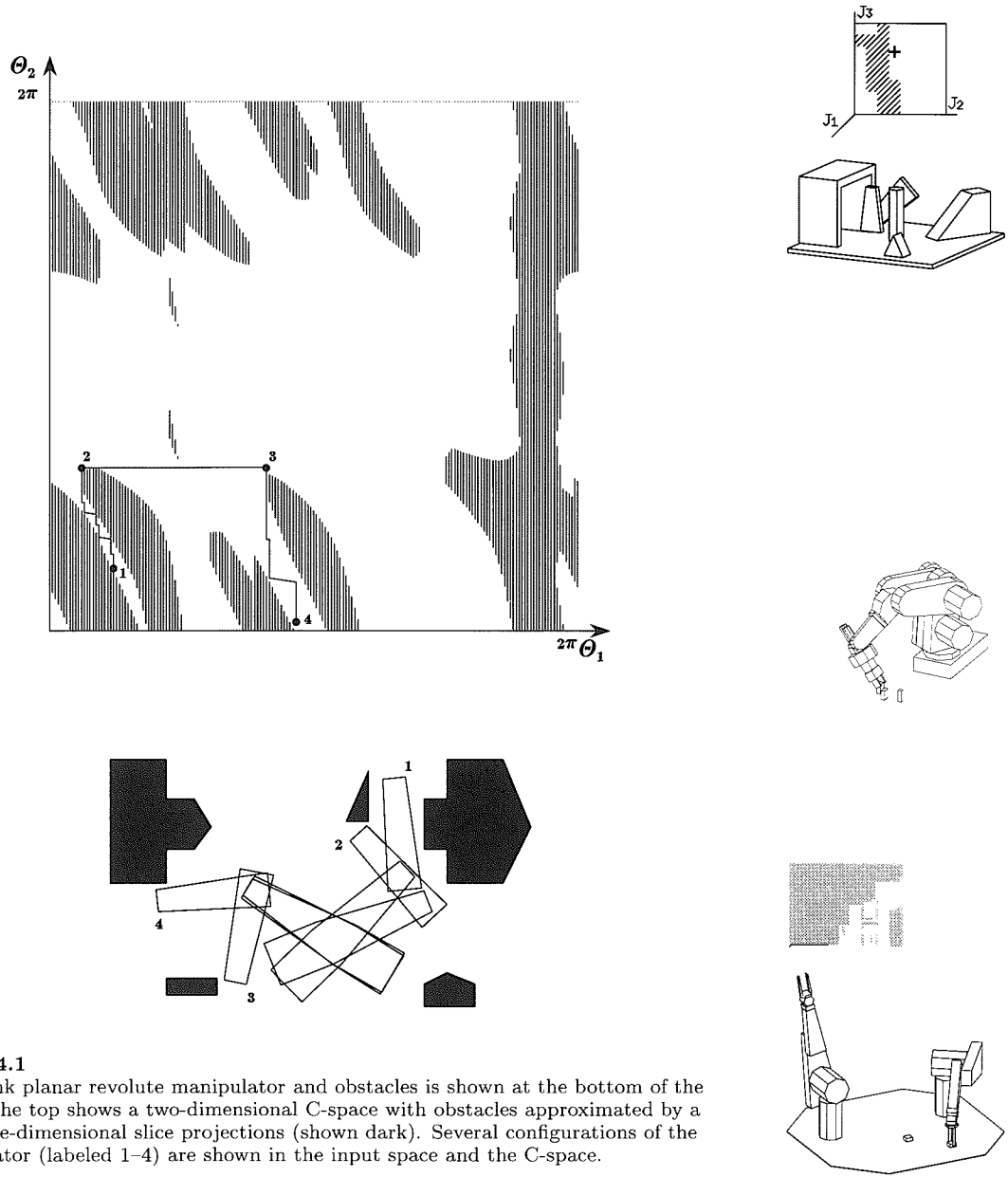
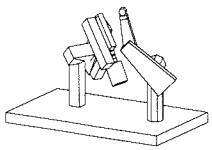


Figure 4.1
A two-link planar revolute manipulator and obstacles is shown at the bottom of the figure. The top shows a two-dimensional C-space with obstacles approximated by a list of one-dimensional slice projections (shown dark). Several configurations of the manipulator (labeled 1–4) are shown in the input space and the C-space.



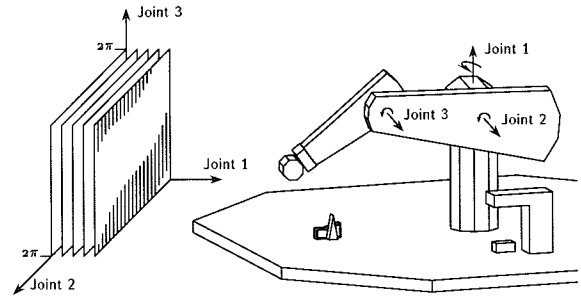
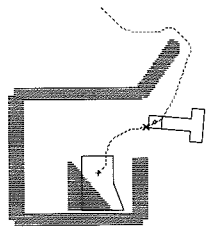


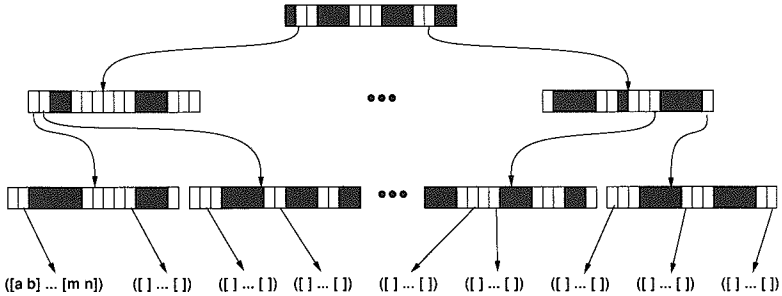
Figure 4.2
Configuration space for the first three links of a Puma robot is represented as a set of two-dimensional slices, each computed for a different value of θ_1.



Observe that the basic computation to be done is that of determining the ranges of legal values for a joint parameter given ranges of values of the previous joints. This is an instance of the slice projection method we first saw in Section 3.4.5.

The recursive nature of the C-space computation calls for a recursive data structure to represent the C-space. One possible implementation uses a tree whose depth is $n - 1$, where n is the number of joints, and whose branching factor is the number of intervals into which the legal joint parameter range for each joint is divided. Figure 4.3 illustrates such a tree. The leaves of the tree are ranges of legal (or forbidden) values for the joint parameter n. Many of the internal nodes in the tree will have no descendants because they produce a collision of some link $i < n$.

The main advantage of a representation method built on recursive slice projection is its simplicity. All operations on the representation boil down to dealing with linear ranges, for which very simple and efficient implementations are possible. The disadvantages are the loss of accuracy, and the rapid increase of storage and processing time with dimensionality of the C-space. Contrast this approach with one that represents the boundaries of the obstacles by their defining equations [9]. Using the defining equations is more compact and more accurate, but the algorithms for dealing with interactions between obstacle boundaries may be very complex.



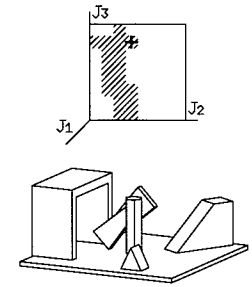
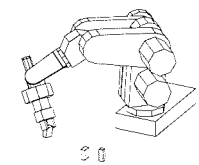


Figure 4.3
The recursive nature of the C-space leads to a recursive data structure: an n-level tree whose leaves represent legal ranges of configurations for the robot manipulator.

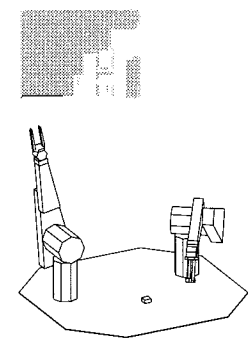
4.1.1 Slice projections for polygonal links

To compute the one dimensional slice projections of C-space obstacles requires determining the range of forbidden values of one joint parameter, given ranges of values for all previous joint parameters. We will illustrate how these ranges may be computed by considering the case of planar revolute manipulators moving among planar obstacles. We will first discuss this problem informally and then derive the solution from the equations of C-surfaces.

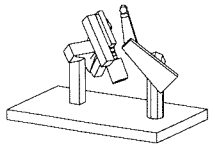


4.1.2 A geometric view

Assume that joint k, a revolute joint, is the free joint parameter for a one-dimensional slice projection and that the previous joints are fixed at known values. We assume, for now, that the previous joints are fixed at *single* values, not *ranges* of values; we will see later how to relax this restriction. We require that the configuration of the first $k-1$ links be safe—that no link intersect an obstacle. This is guaranteed by the recursive computation we saw above. Given these assumptions, we need to find the ranges of values of the single joint parameter θ_k that are forbidden by the presence of objects in the workspace. That is, the range of θ_k for which the k^{th} link polygon collides with some obstacle.



The key geometric observation is that collision between two polygons exists either when an edge of one crosses an edge of the other or when one polygon is completely contained within the other. Furthermore, the



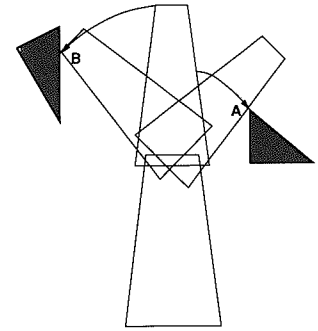
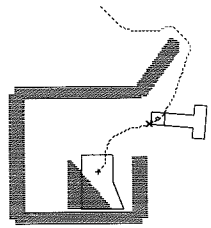


Figure 4.4
Contact conditions for computing one-dimensional slice projections: (a) Vertex of obstacle and edge of link (b) vertex of link and edge of obstacle. The circles indicate the path of the vertices as the link rotates around the specified joint.

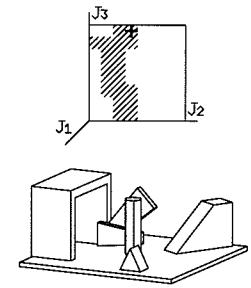


only way a collision can first come into being is by one edge crossing another edge. Since we are interested in avoiding collisions in the first place—not in detecting them after they have happened, we will focus only on edge/edge interactions. Then, determining forbidden ranges of θ_k boils down to computing the union of the angle ranges for which an edge of the link crosses an edge of an obstacle polygon. Because we have ignored the complete overlap case, the absence of a forbidden range for a joint angle does not mean there can be no collision for any joint value; it means there can be no *change* in the collision state.

For two polygon edges that start out disjoint to come to overlap, they must first pass through a vertex/edge contact. Therefore, the ranges of forbidden values for θ_k will be bounded by angles where link k is just touching an obstacle. For polygonal links moving among polygonal obstacles, the extremal contacts happen when a vertex of one object is in contact with an edge of another object. These correspond to the Type A and Type B edges we saw in Section 3.4.6. Therefore, the first step in computing the forbidden ranges for θ_k is to identify those values of θ_k for which some obstacle vertex is in contact with a link edge or some link vertex is in contact with an obstacle edge (Figure 4.4). We call these values **contact angles**.

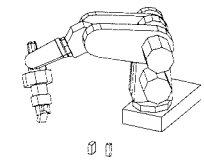
The link is constrained to rotate about its joint, therefore every point on the link follows a circular path when the link rotates. The link

vertices, in particular, are constrained to known circular paths. The intersection of these paths with obstacle edges determine some of the potential contact angles of θ_k, for example, B in Figure 4.4. As the link rotates, the obstacle vertices also follow known circular paths relative to the link. The intersection of these circles with link edges determine the remaining potential contact angles for θ_k, for example, A in Figure 4.4.

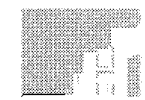


Determining whether a vertex and an edge segment can intersect requires first intersecting the circle traced out by the vertex and the infinite line supporting the edge to compute the potential intersection points. The existence of such an intersection is a *necessary* condition for a contact between link and obstacle, but it is not *sufficient.* In addition, the intersection point must be within the finite edge segment, not just the line supporting the edge. The potential contact angles satisfying this **in-edge constraint** are the actual contact angles.

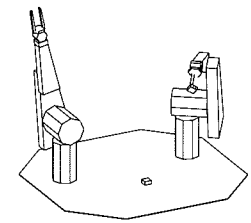
The in-edge constraint can be trivially tested given the potential contact point and the endpoints of the contact edge. Since we know that the contact point is on the line of the edge, all that remains to be determined is whether it lies between the endpoints of the edge. This can be done by ensuring that the x- and y-coordinates of the contact point are within the range of x- and y-coordinates defined by the edge endpoints. Note that for contacts involving link edges and obstacle vertices, the position of the endpoints of the link edge must be rotated around the joint position by the computed value of the joint angle at the contact.



The set of contact angles represent the critical angles at which edges of the link and obstacle initiate or break contact. We can pair up these angles to determine the range of angles for which any pair of link and obstacle edges overlap. The forbidden angle ranges for θ_k are simply the union of these ranges for all edge pairs.

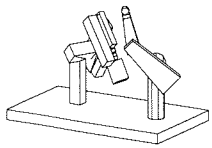


4.1.3 Derivation using C-surfaces



The two types of contacts (obstacle vertex with link edge and obstacle edge with link vertex) give rise to the two basic types of C-space surfaces, type A and type B. The equations of such surfaces are parameterized by the configuration parameters, θ_i.

For a revolute joint, choose the coordinate system to be located at the joint. The coordinate representation of all of the vectors will be relative to this coordinate system; this simplifies some of the derivations.



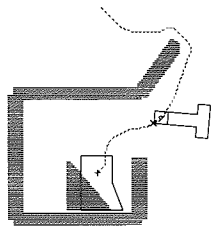
We represent a line supporting an edge by an equation of the form: $\hat{\mathbf{n}} \cdot \mathbf{x} + d = 0$, where $\hat{\mathbf{n}}$ is the (outward pointing) unit vector that is normal to the line and $|d|$ is the perpendicular distance to the edge from the origin. The condition for a vertex $\mathbf{v}$ being in contact with such a line is simply $\hat{\mathbf{n}} \cdot \mathbf{v} + d = 0$.

For a type B contact, we are given a link vertex whose initial position vector (for $\theta_k = 0$) is $\mathbf{v}$ and an obstacle edge whose line equation is $\hat{\mathbf{n}} \cdot \mathbf{x} + d = 0$. If the link angle is θ_k, the coordinates of the rotated link vertex are:

$$\mathbf{v}' = (v_x \cos\theta_k - v_y \sin\theta_k, v_x \sin\theta_k + v_y \cos\theta_k)$$

Substituting into the line equation yields a simple trigonometric equation in θ_k (all the other terms are constant):

$$(n_x v_x + n_y v_y)\cos\theta_k + (n_y v_x - n_x v_y)\sin\theta_k + d = 0 \tag{4.1.1}$$



From the definition of the scalar and vector product, we have that

$$n_x v_x + n_y v_y = \|\mathbf{v}\|\cos\phi, \qquad n_y v_x - n_x v_y = \|\mathbf{v}\|\sin\phi$$

where ϕ is the angle between $\hat{\mathbf{n}}$ and $\mathbf{v}$. From this, it is clear that the C-surface equation is merely

$$\|\mathbf{v}\|\cos(\theta_k - \phi) = -d$$

The solution to this equation is:

$$\theta_k = \cos^{-1}\left(\frac{-d}{\|\mathbf{v}\|}\right) + \phi \tag{4.1.2}$$

Figure 4.5 illustrates this situation. There is one such C-surface for each combination of link vertex and obstacle edge.

Using the same notation, except that the edge is a link edge and the vertex an obstacle vertex, the equation for a type A C-surface is

$$(n_x v_x + n_y v_y)\cos\theta_k - (n_y v_x - n_x v_y)\sin\theta_k + d = 0 \tag{4.1.3}$$

The only difference is the sign of the coefficient of $\sin\theta_k$. This difference arises from the fact that we are thinking of the obstacle vertex as counter-rotating while the link stands still. That is, the direction of rotation of the vertex is the opposite of θ_k; this changes the sign of the sine of the angle. The solution to this equation is:

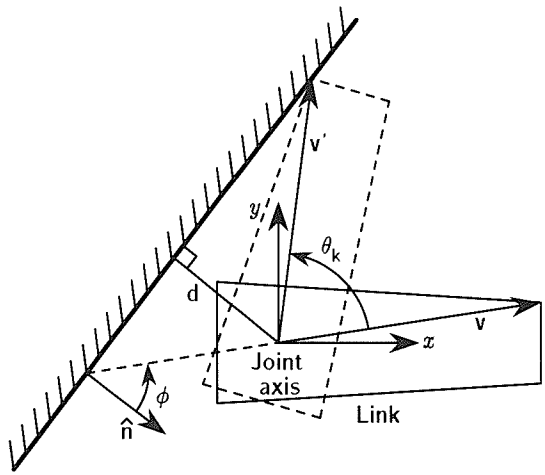


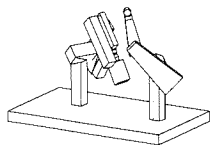
Figure 4.5
Illustration of terms used in Equations (4.1.1) – (4.1.4)

$$\theta_k = \cos^{-1}\left(\frac{-d}{\|\mathbf{v}\|}\right) - \phi \tag{4.1.4}$$

There is one such C-surface for each combination of obstacle vertex and link edge.

Note that there are generally two solutions to each of the equations (arising from the arccosine) since they correspond to intersections of a circle traced out by a vertex and an infinite line supporting an edge. Of course, when the magnitude of the argument to the arccosine is greater than one, this indicates an infeasible contact, that is, the line is beyond the reach of the vertex. (Of course, when the argument is equal to one, the line is tangent to the circle—a grazing contact.) The valid intersections of the circle and line, however, do not necessarily represent contacts between the link and an obstacle. The in-edge constraint must also checked, as described before, by computing the coordinates of the intersection point and the positions of the edge endpoints, given the computed values of θ_k.

Each contact angle computed above can be tagged by the edges involved. Each vertex is the endpoint of two edges, so each contact angle represents a transition point for two edge pairs: each vertex edge paired



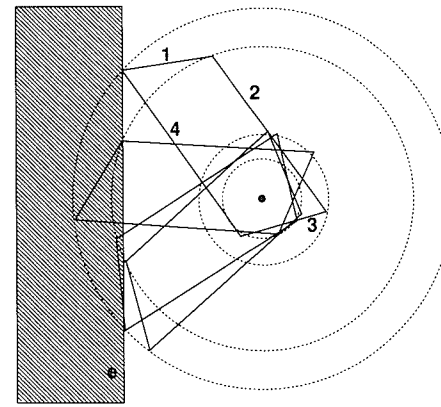
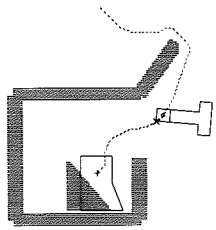


Figure 4.6
Each edge/edge interaction can give rise to 0, 1, or 2 disjoint ranges of angles. In this example, the four positions of the link (the parallelogram) are shown colliding with the rectangular obstacle. The link edge labeled 1 forms two angle ranges with the obstacle edge labeled e; there are two disjoint angle ranges for which there is a collision. The link edges labeled 2 and 4 each have a single collision range with the obstacle edge e. The edge labeled 3 has no collision range with e.



with the contact edge. A given pair of edges can give rise to zero, one or two angle ranges (see Figure 4.6). This follows from the fact that each vertex/edge interaction can give rise to up to two contact angles.

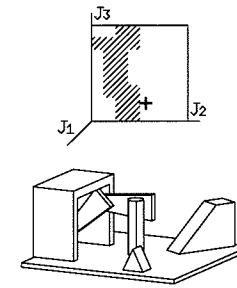
Having constructed the angle ranges for each edge pair, we can combine them to construct the final forbidden range for the joint angle. For example, in Figure 4.6, the union of the ranges arising from the $e:1$, $e:2$ and $e:4$ collisions give rise to the total collision range for the joint.

4.1.4 The effect of ranges of joint angles

Our discussion thus far has been limited to situations where all the joints except the last have known fixed values. The definition of one-dimensional slice projections allows all the joints, save one free joint, to be within a range, not just a single value. We can readily convert the slice projection problem (for ranges of joint values) to the simpler cross section projection problem (for single joint values) we have already discussed.

The key idea is to grow either the link or the obstacles so that any safe placement of the "new" link among the "new" obstacles represents a

range of legal displacements (within the slice joint ranges) of the original link among the original obstacles. There are two approaches that achieve this goal:

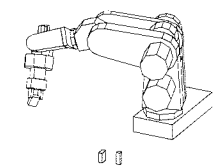


1. One can grow the link by the upper bound on the largest Cartesian displacement of any point on the link in response to a displacement within the specified range of joint values.
2. One can replace the obstacles by the volume they would sweep out if they were attached to the manipulator while it moves within the slice joint ranges.

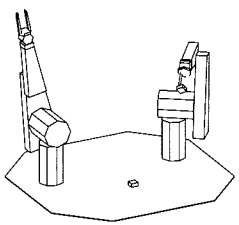
For a detailed treatment of this issue, see [47]. HANDEY attempts to side-step this problem by using a fine enough resolution in its slice projections.

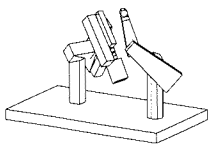
4.2 Slice projections for polyhedral links

The basic approach described in Section 4.1 carries over directly to three-dimensional manipulators and obstacles. The only significant difference is that, since we are dealing with polyhedra, we have slightly different conditions for collision. In the planar case, the relevant collision condition involved the crossing of a link edge and an obstacle edge. Therefore, the beginning and end of a collision are signaled by a vertex/edge transition. In the solid case, collisions involve an edge of one polyhedron crossing a face of another. Therefore there are three collision transitions: type A, vertex of obstacle and face of link; type B, vertex of link and face of obstacle; and type C, edge of link and edge of obstacle (see Section 3.4.4). As in the planar case, the final range of forbidden angles for the joint will be the union of the forbidden ranges for each edge/face pairing.



Let us consider type B contacts first. Each revolute joint is characterized by an axis of rotation. As the joint rotates, link vertices trace circles in a plane whose normal is the joint axis. The intersection of this circle with the plane supporting an obstacle face defines two candidate points of contact. As in the two-dimensional case, possible contacts must satisfy the in-face constraint—the contact must be within the obstacle face. This constraint can be checked using any of many existing algorithms for testing whether a point is in a polygon.





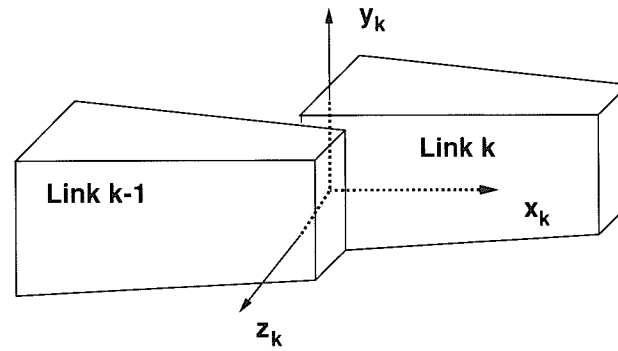
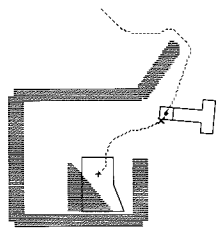


Figure 4.7
Joint coordinate system.



Type A contacts are handled analogously to type B contacts except that now the vertex belongs to an obstacle and the face to a link. The axis of rotation is still that of the manipulator joint.

Detecting type C contacts requires detecting the intersection of a line (supporting a link edge) rotating about the joint axis and a stationary line (supporting an obstacle edge). Of course, an intersection point must be inside both edge segments to be feasible.

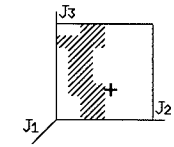
In what follows we assume that the coordinate system is chosen so that the origin corresponds to the position of revolute joint k and the z-axis is aligned with the joint axis (see Figure 4.7). The coordinate representation of all vectors is relative to this coordinate system. We assume that the initial position of the link polyhedron corresponds to $\theta_k = 0$. We are interested in computing values of θ_k for which the link is in contact with the obstacle polyhedron.

4.2.1 Type B contact: link vertex and obstacle face

We are given a vertex of the link whose position vector is $\mathbf{v}$ and an obstacle face whose plane equation is $\hat{\mathbf{n}} \cdot \mathbf{x} + d = 0$ ($\hat{\mathbf{n}}$ is the plane's outward-facing unit normal). We solve for the angle θ_k that rotates the vector $\mathbf{v}$ onto the plane. We obtain the equation for θ_k by substituting the vertex's position, rotated by θ_k, into the plane equation and solving for θ_k.

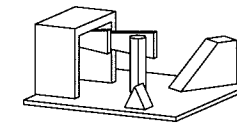
The coordinates of the position vector for the rotated vertex are:

$$\mathbf{v}' = (v_x \cos\theta_k - v_y \sin\theta_k, v_x \sin\theta_k + v_y \cos\theta_k, v_z)$$



Substituting into the plane equation gives $\hat{\mathbf{n}}\cdot\mathbf{v}'+d = 0$, yielding a simple trigonometric equation

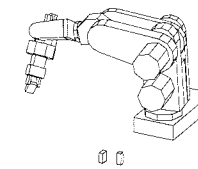
$$(n_x v_x + n_y v_y)\cos\theta_k + (n_y v_x - n_x v_y)\sin\theta_k = -d - n_z v_z \tag{4.2.5}$$



whose solution is:

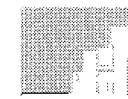
$$\begin{aligned}\theta_k &= \cos^{-1}\left(\frac{-n_z v_z - d}{\sqrt{(v_x^2+v_y^2)(1-n_z^2)}}\right) \\ &\quad + \operatorname{atan}(n_y v_x - n_x v_y, n_x v_x + n_y v_y)\end{aligned} \tag{4.2.6}$$

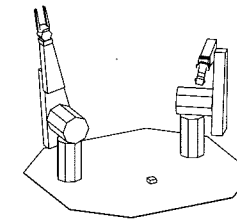
Eq. (4.2.5) is the equation for a type B C-surface. Note that if we let $n_z = 0$, then Eqs. (4.2.5) and (4.1.1) are essentially identical. The arctangent simply computes the angle between the plane normal and the projection of $\mathbf{v}$ on the xy-plane; this magnitude is analogous to ϕ in the planar case. Eqs. (4.2.6) and (4.1.2) are also related in the same way.



4.2.2 Type A contact: obstacle vertex and link face

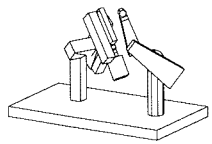
We are given an obstacle vertex whose position vector is $\mathbf{v}$ and a link face whose plane equation is $\hat{\mathbf{n}}\cdot\mathbf{x}+d = 0$ ($\hat{\mathbf{n}}$ is the plane's outward-facing normal). The solution for θ_k is almost identical to the type B case, the only difference is the sign of the first argument to the arctangent. This reflects the fact that in a type A contact we are treating the link as stationary and assuming the object is rotating *in the opposite direction.* This changes the sign of the sine of the angle.





$$\begin{aligned}\theta_k &= \cos^{-1}\left(\frac{-n_z v_z - d}{\sqrt{(v_x^2+v_y^2)(1-n_z^2)}}\right) \\ &\quad + \operatorname{atan}(n_x v_y - n_y v_x, n_x v_x + n_y v_y)\end{aligned} \tag{4.2.7}$$

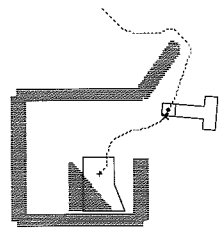
4.2.3 Type C contact: obstacle edge and link edge



This case is substantially more difficult. We represent points on the edges parametrically in t. Therefore, points on the link's edge are represented by $t_l\mathbf{m}+\mathbf{v}$ where $\mathbf{v}$ is the position vector of one of the endpoints of the edge and $\mathbf{m}$ is a vector along the edge (actually the difference vector between the endpoints). The parameter $t_l \in [0,1]$ parameterizes position along the edge. We can represent the vector along the obstacle edge similarly as $t_o\mathbf{n}+\mathbf{w}$ for $t_o \in [0,1]$.

As the edge rotates around the z-axis, points on the edge trace out circles. The equation for points on those circles are:

$$\begin{aligned} x^2+y^2 &= (m_x t_l + v_x)^2 + (m_y t_l + v_y)^2 \\ z &= m_z t_l + v_z \end{aligned}$$



These can be combined by solving the second equation for $t_l = (z - v_z)/m_z$ and substituting into the first to obtain:

$$x^2+y^2 = \left(\frac{m_x}{m_z}(z-v_z)+v_x\right)^2 + \left(\frac{m_y}{m_z}(z-v_z)+v_y\right)^2 \qquad (4.2.8)$$

This is an implicit equation for points on the rotation surface.

The parametric form of the obstacle edge can be used to solve for the intersection of the edge with the rotation surface.

$$x = n_x t_o + w_x, \qquad y = n_y t_o + w_y, \qquad z = n_z t_o + w_z$$

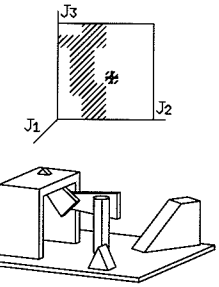
Substituting into (4.2.8) gives a quadratic equation in t_o.

Define the following terms:

$$\begin{aligned} p &= (n_x^2+n_y^2)m_z^2 - (m_x^2+m_y^2)n_z^2 \\ q &= 2[(n_x w_x + n_y w_y)m_z^2 - (m_x^2+m_y^2)(w_z-v_z)n_z \\ &\quad - (m_x v_x + m_y v_y)m_z n_z] \\ r &= (w_x^2+w_y^2)m_z^2 \\ &\quad - [(m_x(w_z-v_z)+v_x m_z)^2 + (m_y(w_z-v_z)+v_y m_z)^2] \end{aligned}$$

The quadratic equation that must be solved for t_o is

$$pt_o^2 + qt_o + r = 0$$



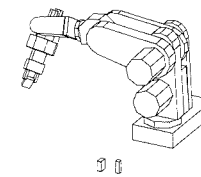
Having t_o we can solve for t_l since we know that the z values at contact must be equal. Therefore,

$$t_l = \frac{n_z t_o + w_z - v_z}{m_z}$$

Given values for t_l and t_o, we must first check that they are in the range $[0, 1]$ (the in-edge constraint), then we can compute points of intersection on each of the edges. Let $\mathbf{l}$ be the position vector of the intersection point on the link edge and $\mathbf{o}$ the position of the intersection point on the object edge. Then,

$$\theta_k = \text{atan}(l_x o_y - l_y o_x, l_x o_x + l_y o_y)$$

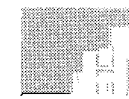
We have assumed, when deriving (4.2.8), that $m_z \neq 0$. In the not uncommon event that, in fact, $m_z = 0$, then all the points on the rotation surface have $z = v_z$ and so will the intersection point with the obstacle edge. We can use this to obtain $t_o = (v_z - w_z)/n_z$. We can then solve for t_l by using the fact that the contact point on the link edge will be on the same circle as the contact point on the obstacle edge:



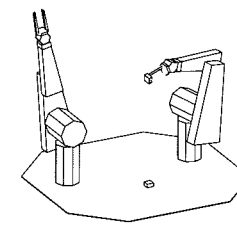
$$(m_x t_l + v_x)^2 + (m_y t_l + v_y)^2 = (n_x t_o + w_x)^2 + (n_y t_o + w_y)^2$$

But we know the value of t_o so this is a quadratic equation for t_l. Given the values of t_l and t_o we can solve for θ_k as above.

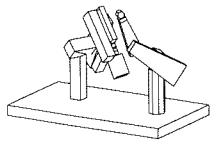
4.2.4 Merging the ranges



Each vertex/face contact determines one end of the range for the contact face paired with each edge that defines the contact vertex. Each edge/edge contact determines one end of the range for the contact edges paired with each of the faces defining the other edge. Having constructed the angle ranges for each edge/face pair, we can combine them to construct the forbidden range for the joint angle. This is completely analogous to the process in the planar case.



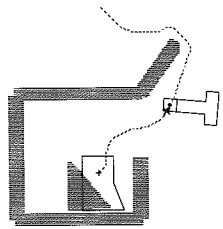
4.3 Efficiency considerations



Thus far, we have focused on the basic algorithmic and geometric details of constructing C-space maps. Our discussion, however, has not addressed two crucial issues:

1. Efficiency: How can we reduce the time required to build C-space maps?
2. Search: How should C-space maps be searched for a path?

The next two sections describe the strategies adopted by the HANDEY system to answer these questions.



The straightforward application of C-space maps to the problem of searching for safe paths for a six-joint manipulator involves building a high-resolution six-dimensional C-space map. This strategy is extremely wasteful, however. The worst-case time to compute a C-space map using the slice projection algorithm described in Section 4.1 grows as

$$r^{k-1} f(n)$$

where:

r is the number of slices computed for each joint,

k is the number of joints in the manipulator, and

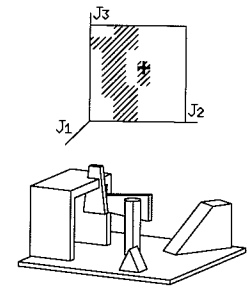
$f(n)$ is the time to compute a single slice as a function of n, a measure of the complexity of the environment such as number of edges in the obstacles.

This time complexity means that increased resolution and increased number of joints rapidly increase the time to compute C-space maps. It is obvious from the form of this complexity measure that we can reduce the actual time spent in computing C-space maps by:

1. reducing the number of joints considered, k,
2. reducing the number of slices, r, or
3. reducing the time to compute a single slice, $f(n)$.

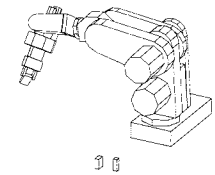
Another useful strategy is to cache intermediate results that may be useful later. HANDEY makes use of all of these strategies.

4.3.1 Reducing the number of joints



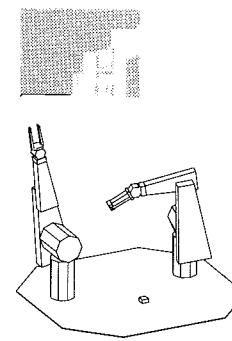
When making a large motion from one location in the workspace to another, it is seldom the case that the orientation of the gripper matters. Recall that the first three joints of a manipulator are sufficient to move the gripper but not to control its orientation. The last three joints, which we will call the **wrist joints**[2], primarily affect the orientation of the gripper. This suggests that when searching for a collision-free path, we consider only motions of the first three joints of the manipulator in detail and pay less attention to the values of the wrist joints. Constructing the resulting three-dimensional C-space maps will be much more manageable than constructing a six-dimensional map. There are, however, two ways of "ignoring" the wrist joints. One way is by holding their values fixed and the other is by considering them as free to move over some range of values. In either case, we are constructing slice projections of the full six-dimensional C-space map.

Let the starting joint angles of a manipulator be denoted $\boldsymbol{\theta}_c$ and the goal joint angles as $\boldsymbol{\theta}_g$. Let $\boldsymbol{\theta}^{[i,j]}$ represent the sequence of joint values $(\theta_i, ..., \theta_j)$. Gross motion planning in HANDEY is carried out using three slices of the six-dimensional C-space:

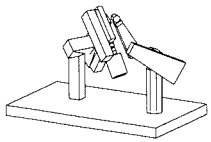


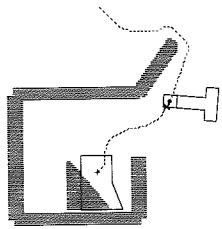
1. A slice constructed with $\boldsymbol{\theta}^{[4,6]} = \boldsymbol{\theta}_c^{[4,6]}$, call it the **start slice.**
2. A slice constructed with $\boldsymbol{\theta}^{[4,6]} \in [\boldsymbol{\theta}_c^{[4,6]}, \boldsymbol{\theta}_g^{[4,6]}]$, call it the **reorientation slice.**
3. A slice constructed with $\boldsymbol{\theta}^{[4,6]} = \boldsymbol{\theta}_g^{[4,6]}$, call it the **goal slice.**

In the start slice, the wrist joints are held fixed at the values they have in the starting configuration of the manipulator. In the reorientation slice, the wrist joint angles are allowed to vary between their values in the start and the goal configurations. To compute this slice, the wrist links and the gripper are replaced by an approximation to their swept volume as they move within this range of values. In the goal slice, the wrist joints are held fixed at the values they have in the goal configuration of the manipulator. Note that any safe point in the reorientation slice must also be a safe point in both the start and goal slices.



[2]Technically, a manipulator wrist exists when the last three joint axes intersect at a common point. This is the most common arrangement for industrial manipulators since it guarantees a closed-form solution to the inverse kinematics [66].





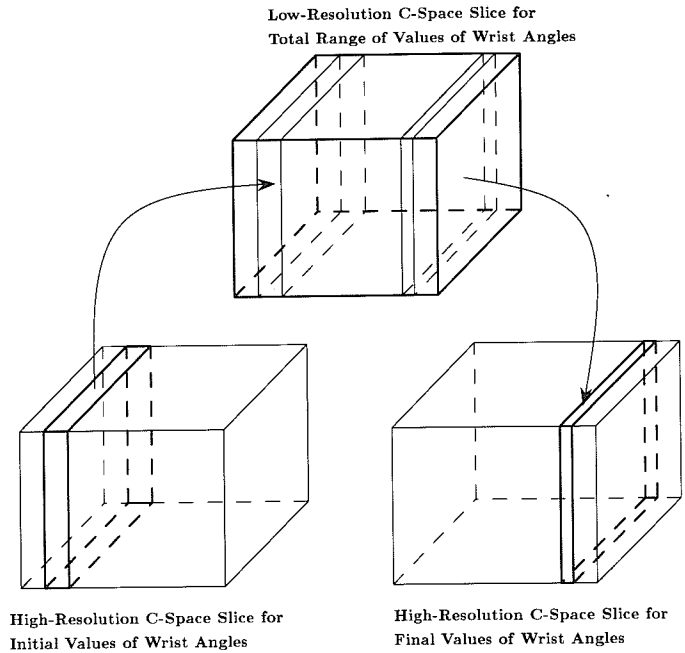
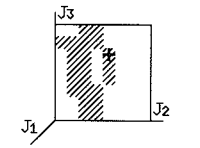


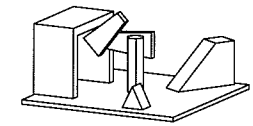
Figure 4.8
The gross motion planning problem is split into three 3-degree-of-freedom slices.

The search problem then involves finding a path that begins at the start joint angles (in the start slice) that moves into the reorientation slice and ends at the goal joint angles (in the goal slice). Figure 4.8 illustrates this approach. Note that we are basically using slice projection to approximate a six-dimensional C-space map by three three-dimensional C-space maps. From a computational viewpoint this is an excellent tradeoff since the time to compute three three-dimensional slices ($3r^2f(n)$) typically will be much less that to compute a single six-dimensional slice ($r^5f(n)$).

We should emphasize that the choice of three three-dimensional slices is based on a tradeoff between accuracy and efficiency. Note that the wider the range of angles used to define a slice, the smaller the free space becomes in that slice, or, equivalently, the farther from obstacles the manipulator must remain. The choice of slices in HANDEY is based on

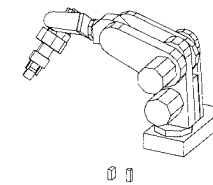
the observation that we want maximal accuracy of representation near the start and goal configurations. The start and goal slices are therefore defined for fixed angles to allow approach and departure motions near obstacles. The reorientation motion for the gripper can (and should) happen away from the obstacles, therefore, we use the whole range of required motion to define the remaining slice.





The use of only three three-dimensional slices to approximate a six-dimensional slice means that there are, in principle, many problems for which paths will not be found and for which paths do, in fact, exist. For example, if the gripper is carrying a long bar or if the workspace is very cluttered, then the orientation of the gripper may matter a great deal and the use of a single reorientation slice will likely lead to failure. We could extend the approach described above to use multiple reorientation slices, defined for small ranges of angles, when the gripped object is large or the space is cluttered, but the current version of HANDEY does not do this.

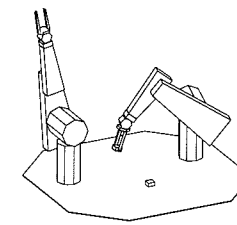
4.3.2 Reducing the number of slices



Using the recursive slice projection algorithm of Section 4.1, the three-dimensional slices of the six-dimensional C-space are approximated by a set of r_2 two-dimensional slices, each of which is in turn approximated by r_1 one-dimensional slices (see Figure 4.3, page 61). In the simplest case of uniform sampling of the total angle range for each joint, we have that $r_2 = r_1 = r$, where $r = (2\pi/\text{angle step size})$. In that case, computing one three-dimensional slice requires computing on the order of r^2 one-dimensional slices.

We can reduce the amount of computation required to compute slices either:

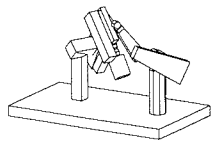
1. by reducing the size of the joint angle range represented in the slice, or
2. by increasing the angle step size by lowering the angular resolution.



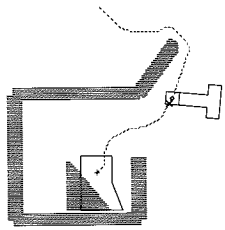
HANDEY uses both of these techniques to advantage.

The number of one-dimensional slices needed to compute a three-dimensional slice can be reduced by using a very low angular resolution in computing the reorientation slice. The rationale is that this slice is already a gross approximation defined for a wide range of angles of the

wrist joints. Furthermore, the working assumption is that the reorientation motion will take place relatively far from the obstacles. Any motion close to the obstacles will take place in the start or goal slices, which are computed using constant values of the wrist joint angles. These slices need to be computed at high resolution.



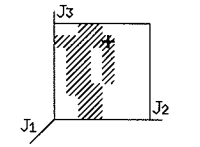
The number of one-dimensional slices needed to compute a three-dimensional slice can also be reduced by building only as much of the C-space map as is needed to find a safe path. When searching for a path from $\boldsymbol{\theta}_c^{[1,3]}$ to $\boldsymbol{\theta}_g^{[1,3]}$, HANDEY first constructs the C-space map only within the bounding box defined by these two joint vectors. If a safe path cannot be found within this subset of the C-space map, then the computed area of the C-space map is broadened iteratively until either a path is found or some pre-determined maximum area of the C-space map is computed. In this approach, the construction of the C-space map proceeds on an "on-demand" basis.

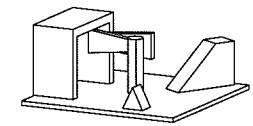


The tricky issue in implementing on-demand construction of the C-space map arises from the fact that the representation of the C-space is split among three slices: start, reorientation, and goal. Each of the three slices covers the range of possible values of the first three joints, but is defined for a different range of values of the wrist joints. The defining wrist joint angles for the start slice are included in the defining wrist angle range for the reorientation slice. This means that a particular joint vector $\boldsymbol{\theta}^{[1,3]}$ in the start slice has a corresponding joint vector in the reconfiguration slice. Note that, in general, points near obstacles may be free in the start or goal slice but inside a C-space obstacle in the reorientation slice. If two corresponding joint vectors are in free space in their respective slices then a safe path can go from one slice to the other through this point. An analogous relationship exists between points in the goal slice and the corresponding points in the reorientation slice.

In order for a path to connect from the start configuration to the goal configuration, there must be a safe path from the specified initial point in the start slice to some safe point in the reorientation slice, a path from there to a point safe both in the reorientation and goal slices, and from that point to the specified final point in the goal slice. These connectivity conditions constrain what portions of the corresponding slices must actually be computed. Minimally:

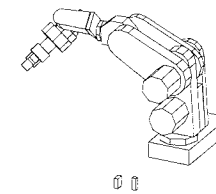
1. The reorientation slice must span the space traced by a line between the start and goal points.
2. The start slice must include the initial point and must overlap the computed area of the reorientation slice.
3. The goal slice must include the goal point and must overlap the computed area of the reorientation slice.





Larger portions of these slices may have to be computed if this minimal subset does not include a safe path, that is, if the free spaces in the slices are not connected. The issue is, however, how to choose between expanding the computed portion of the start, goal, or reorientation slices?

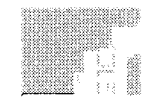
The approach employed in HANDEY is the following: First, compute the minimal set described above. If a path is found in this subset, we are done. Otherwise, compute the complete reorientation slice at low resolution and then expand the start and goal slices just enough so that free space in those slices connects with one of the connected components of the free space in the reorientation slice. In practice, this means computing one or more one-dimensional slices, testing connectivity among the free spaces and repeating. This approach is predicated on the premise that computing the low-resolution reorientation slice can be done much more efficiently than the detailed start and goal slices. We will see in the next section that this is generally the case.

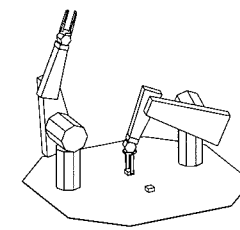


4.3.3 Reducing the time to compute a single slice

The time to compute a single one-dimensional slice using the approach in Section 4.1 depends on two factors:

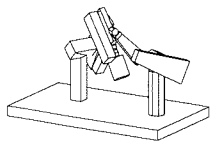
1. The complexity of the polyhedral description of the links and obstacles.
2. The dimensionality of the links and obstacles.



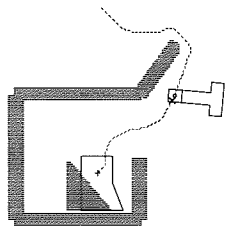


Clearly, when moving far away from an obstacle, its detailed surface features are irrelevant. We do not want to spend time computing detailed C-space obstacles for very complex objects lying in the workspace away from the start and goal points; using a bounding box or sphere approximation is usually sufficient in these cases. The key issue in implementing this strategy is deciding when it is safe to use the approximation. HANDEY uses a very simple strategy: use bounding boxes for everything when computing the reorientation slice and no approximations when

computing the start and goal slices. This strategy is compatible with the fact that HANDEY already tries to minimize the number of slices in the start and goal slices.



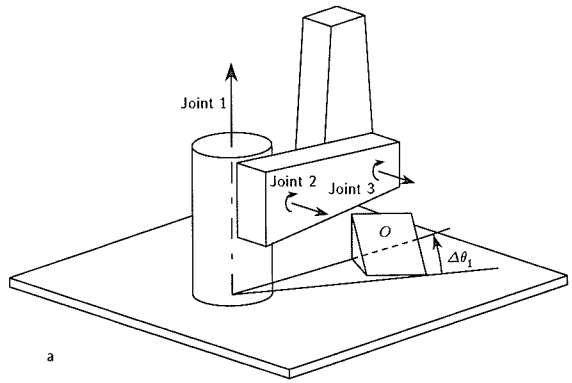
The time to compute a single one-dimensional slice for planar links and obstacles grows as $v_l e_o + v_o e_l$ where v denotes numbers of vertices and e numbers of edges; the subscript l indicates the link description and o indicates the obstacle description. For solid links and obstacles, the time grows as $v_l f_o + v_o f_l + e_l e_o$, where f denotes numbers of faces. Clearly, the simplicity of polygonal descriptions when compared to polyhedral descriptions means that computing one-dimensional slices for planar links and obstacles will take much less time on the average. This is borne out in practice. The moral is that we should use planar approximations whenever possible.

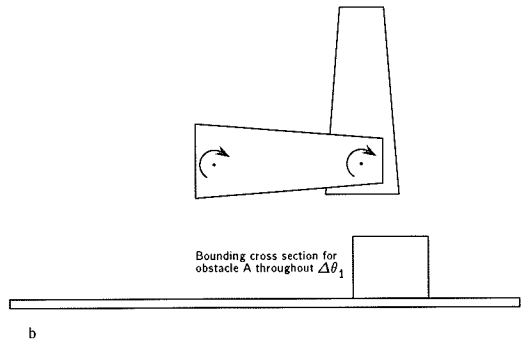


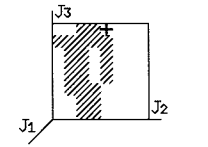
We now show how low-resolution C-space maps can be efficiently constructed using primarily planar computations. Many commercial revolute robots have two adjacent revolute joints with parallel rotation axes, for example, joints 2 and 3 of the Puma, preceded by another revolute joint, for example, joint 1 of the Puma, whose axis is orthogonal to those two (see Figure 4.9(a)). This means that we can approximate the manipulator as a planar two-link arm operating in a plane determined by the first joint angle. Note that many apparently dissimilar robots such as the Unimation Puma, Asea, Hitachi, Yasukawa Motoman, Minimover, and Rhino share this property.

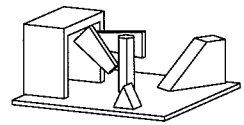
Assume a fixed value, θ_1, for the first joint of one of these manipulators. Now, consider a plane $P(\theta_1)$ orthogonal to the joint axes of joints two and three and located as shown in Figure 4.9. As θ_1 changes, this plane cuts the obstacles in the workspace. For any obstacle O, we can find the bounding box of all the cross sections of O produced by all values of θ_1. This bounding box can serve as the obstacle for a planar manipulator whose link shapes are defined by the projection of links 2 and 3 onto the plane. Effectively, we are approximating the actual obstacles by toroids, centered at the base of the robot, with constant rectangular cross sections in the plane of the second and third links of the robot (see Figure 4.9(b)).

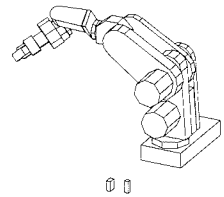
A low-resolution C-space map for the first three joints of the robot can be readily computed by a simple variant of the slice projection algorithm described earlier. Because we have chosen to approximate the obstacles by toroids of constant cross section over their range of θ_1, we will not











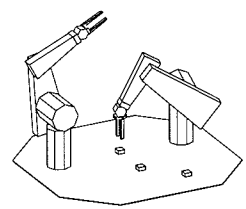
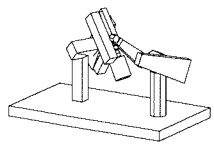
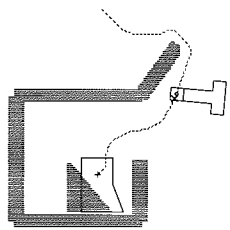


Figure 4.9
Obstacles can be approximated by toroids, centered at the base of the robot, with constant rectangular cross sections in the plane of the second and third links of the robot. (a) A sample world with obstacles: the table and a box on the table. (b) The rectangular cross sections of the obstacles used by the planner



need to sample θ_1 at a fixed resolution. Instead, we only need to sample θ_1 at those critical values where an obstacle ends or a new one starts. Sampling at any other values would produce a map of obstacles identical to one at a critical value of θ_1. Note that every combination of obstacles encountered in one of these slices will simply be the union of some subset of the original obstacles. Since the C-space map of a union is simply the union of the C-space maps (Section 3.4.3) it suffices to compute the C-space maps, for joints 2 and 3, for the rectangular cross section of each approximate obstacle. The C-space maps for the complete slices can then be constructed by a simple union of the C-space maps of each obstacle in the slice. Furthermore, these obstacle maps can be stored in an obstacle cache for the next time the gross-motion planner is called, since most of the obstacles will not have changed.



In the description above we have ignored the width of the links and any offset between them in defining the obstacle cross sections. In practice, we need to take this effective width into account when computing the bounding box. The details are messy but straightforward. Other types of robot kinematics can also be approximated as planar manipulators revolving about the axis of the first joint. The only difference is the type of planar manipulator produced by the approximation.

4.4 Searching for paths

We have seen that the C-space representation that arises naturally out of the recursive slice projection algorithms described above is an n-level tree whose leaves represent legal ranges of configurations for the robot manipulator (see Figure 4.3, page 61). The remaining question is how to search for a collision-free path in a configuration space represented by such a tree. HANDEY uses a very simple search strategy.

The basic idea is to construct a **free-space graph** whose nodes correspond to legal ranges at the leaves of the C-space tree and where a link between nodes indicates that the ranges are adjacent in the configuration space. In HANDEY, adjacency is defined as: reachable by a unit change in exactly one joint parameter, where a unit change is defined as moving between neighboring slices for that joint parameter. This approach is illustrated in Figure 4.10. In a two-dimensional C-space array, this is analogous to having the cells be 4-connected instead of

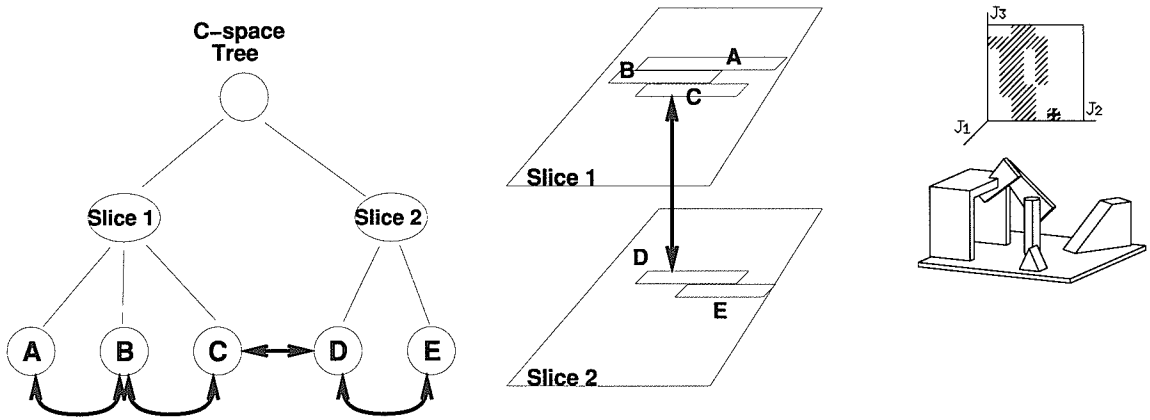
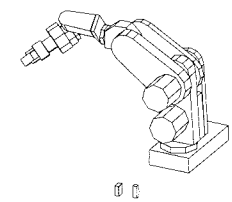
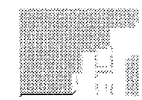


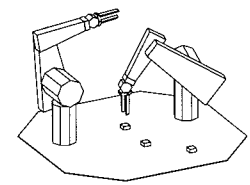
Figure 4.10
A three-dimensional C-space tree with connectivity between free ranges indicated. The free-space ranges corresponding to that tree with the connection between ranges C and D, which go between slices, indicated.



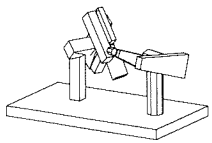
8-connected. The motivation for this choice is efficiency. The number of neighbors of a tree leaf using our definition of adjacency grows linearly with the dimensionality of the C-space. If we were to define adjacency as reachability by a unit change in one or more of the joint parameters, then the numbers of neighbors of a leaf would grow exponentially with the dimensionality of the C-space.

The free-space graph is searched using a standard shortest path algorithm. The definition of distance in the free-space graph is, unfortunately, problematic. If the graph nodes corresponded to points in the C-space, there would be well-defined notions of the length of a link connecting two nodes, using, for example, Manhattan or Euclidean metrics. But, in our free-space graph, a node corresponds to a range of configurations (line segments in the C-space). Therefore, there are an infinite number of paths that move from a range to an adjacent one. The actual shortest path through a given set of ranges cannot readily be determined incrementally as the search proceeds.





HANDEY, therefore, uses a simple local approximation to the actual length of the best path through a set of ranges. Every graph-search algorithm consists of a search in the space of partial paths through the



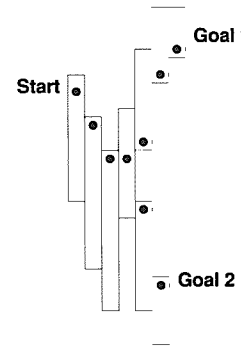
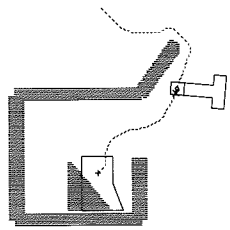


Figure 4.11
The rectangles represent ranges of free values of, say, θ_2 for a given value of, say, θ_1. The graph search procedure keeps track of a position (shown as a filled circle) associated with the last node of a partial path. This position is used to define a unique distance between ranges. The track of positions used to reach two different goals from a given start are shown.

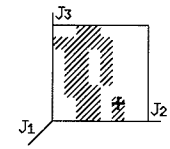


graph. At any point in time, such an algorithm keeps track of a set of partial paths that are eligible for expansion. In our free-space search, a partial path is merely a list of leaves in the C-space tree. In HANDEY, the search algorithm keeps track of not only the partial paths but also of a specific position in the range at the end of the partial path (see Figure 4.11). This position is chosen to be the point closest to the previous range in the partial path. HANDEY uses the closest distance between the end position associated with a partial path and a new graph node as a measure of the distance between two nodes. The resulting paths are not optimal, but they are generally reasonable. One could improve the paths by defining a larger set of connection points between adjacent ranges and searching for shortest paths through these points.

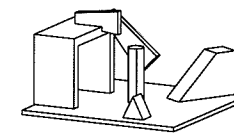
4.5 A massively parallel algorithm

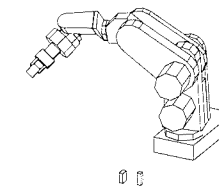
The motion planner described in the preceding sections is intended to operate on a traditional serial computer. In this section, we present an alternative motion planning algorithm that is both practical on sequential machines and can take advantage of the capabilities of **massively-parallel computers**. It is simple enough that it could be implemented

in hardware, potentially leading to real-time configuration space path planning. Also, the description of the algorithm illustrates several important features of the configuration space representation of obstacles.



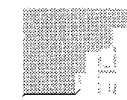
The key idea that makes this algorithm possible was first articulated in W. Newman's doctoral thesis [56]. Newman noted that, for most robot kinematics, one can precompute a one-parameter family of **primitive configuration space maps**, indexed only by radial distance to the robot base. One can then obtain complex C-space maps by superposition of these primitive obstacle maps. This result is a generalization of the union property of the configuration space, where the configuration space obstacle for a union of objects is the union of the configuration space obstacles of the individual objects [46]. (See Section 3.4.3.) The superposition of primitive maps is quite general and can be used to construct C-space maps for the first three links of the majority of existing industrial robots. We have exploited an extension of this property to develop parallel algorithms for computing the C-space maps for six-degree-of-freedom wrist-decoupled robots.

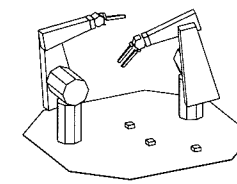




The primitive C-space maps can be represented as **bitmaps**. The advantage of a bitmap representation is that it provides a simple and efficient means of superimposing precomputed maps. Furthermore, the bitmap representation of obstacles can readily be adapted to interface to a wide variety of distance sensors, such as laser range-finders and sonar. The drawback of a bitmap representation is that the memory requirements limit the practical resolution of the map representation, and it forces a discretization of the configuration space in all dimensions.

We have implemented this motion planner for the first three degrees-of-freedom of a Puma robot in *Lisp on a Thinking Machines Corporation's Connection Machine with 8192 processors [28]. The time to build a three-dimensional configuration space quantized at 11 degrees is approximately 0.3 sec; for a configuration space with twice that resolution, the time is approximately two seconds. In both cases, the running time is independent of the number of obstacle bits of the input obstacle maps. Increasing the number of physical processors in the Connection Machine, which can have up to 64K processors, would provide a linear speedup.

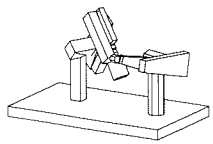




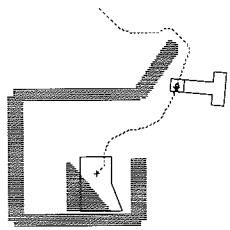
We have also implemented a six-degree-of-freedom version of the algorithm. This algorithm performs a sequential search of the six-dimensional configuration space, building three-dimensional slice projections

in parallel. In the example illustrated in Figure 4.12 a path was found in approximately three minutes.

4.5.1 Primitive configuration space maps



The algorithm is based on exploiting the rotational symmetry in the kinematic structure of most robots to allow precomputation of the configuration space maps for simple obstacles [5, 56]. We will illustrate this idea first for a two-link planar revolute robot, then for the first three links of a Puma-like robot, next for the first three links of a variety of other robot types, and finally for wrist-decoupled six-degrees-of-freedom robots. The treatment of the first three links of robots parallels that in [5, 56]; the treatment of six-degrees-of-freedom robots is new. In this section we describe the general approach. Section 4.5.2 generalizes the approach to six-degree-of-freedom grippers. In Section 4.5.4 we show the actual serial algorithms which implement the approach. In Section 4.5.6 we show the parallel algorithms.



A two-link revolute robot Consider the two-link planar manipulator shown in Figure 4.13. The configuration of this manipulator is specified by the two joint angles θ_2, θ_3. The choice of joint angle subscripts 2 and 3, instead of 1 and 2, is to retain consistency with the three-dimensional robot. In Figure 4.13, we can also see a point obstacle whose polar coordinates are r, ϕ. The C-space obstacle corresponding to collisions between this point obstacle and the last link is shown in the bottom of Figure 4.13. This configuration space obstacle is made up of a single locus, described by the following parametric equations with parameter s (which is the distance on the second link where the point intersects):

$$\theta_3 = \cos^{-1}\left(\frac{r^2 - l_2^2 - s^2}{2l_2 s}\right) \tag{4.5.9}$$

$$\theta_2 = \phi - \text{atan}(s\sin\theta_3, l_2 + s\cos\theta_3) \tag{4.5.10}$$

Or, schematically:

$$\theta_3 = f_3(r, s) \tag{4.5.11}$$

$$\theta_2 = \phi + f_2(r, s) \tag{4.5.12}$$

The crucial observation is that ϕ enters into these equations only as an offset in θ_2. The only obstacle parameter that affects the shape of

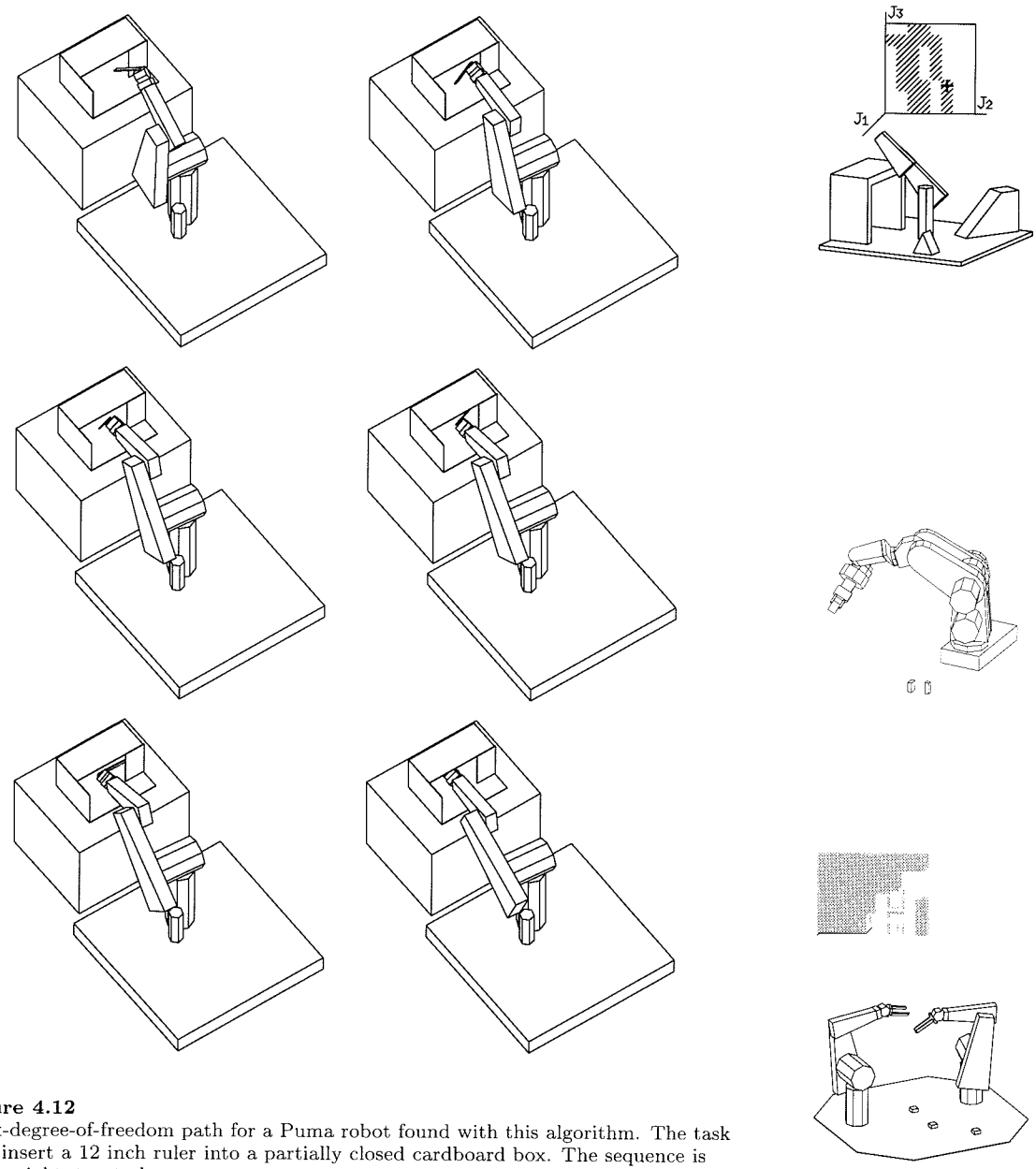
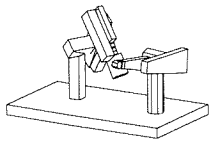
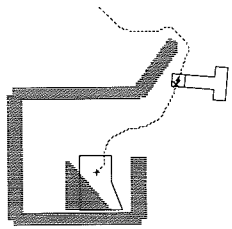


Figure 4.12
A six-degree-of-freedom path for a Puma robot found with this algorithm. The task is to insert a 12 inch ruler into a partially closed cardboard box. The sequence is left to right, top to bottom.



the locus is r. The C-space obstacle for another point with the same r but different ϕ can be derived from the one in Figure 4.13 simply by shifting in the θ_2 direction. Note that when $r \leq l_2$, all configurations for which $\theta_2 = \phi$ are forbidden because of a collision of the first link with the point obstacle. But, once again, ϕ enters only as an offset in θ_2.

This property follows from the rotational symmetry of the problem; the choice of the zero value for θ_2 is totally arbitrary. Had we chosen the base frame to be rotated so that the ϕ coordinate for the obstacle point was zero, we would not have changed the geometry of the problem in any way. A rotation of the base coordinate system amounts to a shift in the configuration space. Therefore, we should expect the effect of a coordinate system rotation to manifest itself as a shift in the configuration space; it does.



Given this property, we can characterize the **primitive configuration space maps** for a given manipulator by storing the configuration space generated from point obstacles with $\phi = 0$ and different values of r. Each of these maps, for this simple manipulator, is characterized by a single locus. The primitive maps form a one-parameter family of curves, $(f_2(r,s), f_3(r,s))$, in the configuration space with radial distance r as a family parameter.

The fact that the shape of the configuration space obstacles for an obstacle point depend only on r is not limited to manipulator links modeled as lines; it holds for any link shape. It follows directly from the rotational symmetry of the problem. Therefore, one can build the primitive configuration space maps for each range of r using any existing configuration space algorithm, or simply by finely sampling the positions of the robot and testing for collisions with the point obstacles. Since this only needs to be done once for a given robot, there is no particular need for efficiency. Figure 4.14 shows some primitive configuration space maps for links with more complex shapes. These maps can be used instead of the obstacle maps for line links shown in Figure 4.13. The extension from point obstacles to physical primitives is trivial, provided the primitive obstacle is symmetric around the relevant joint axis of the robot.

The third dimension The approach which we have used to handle rotation about the base in the planar case can also be applied to rotation about the shoulder axis of a three-dimensional robot.

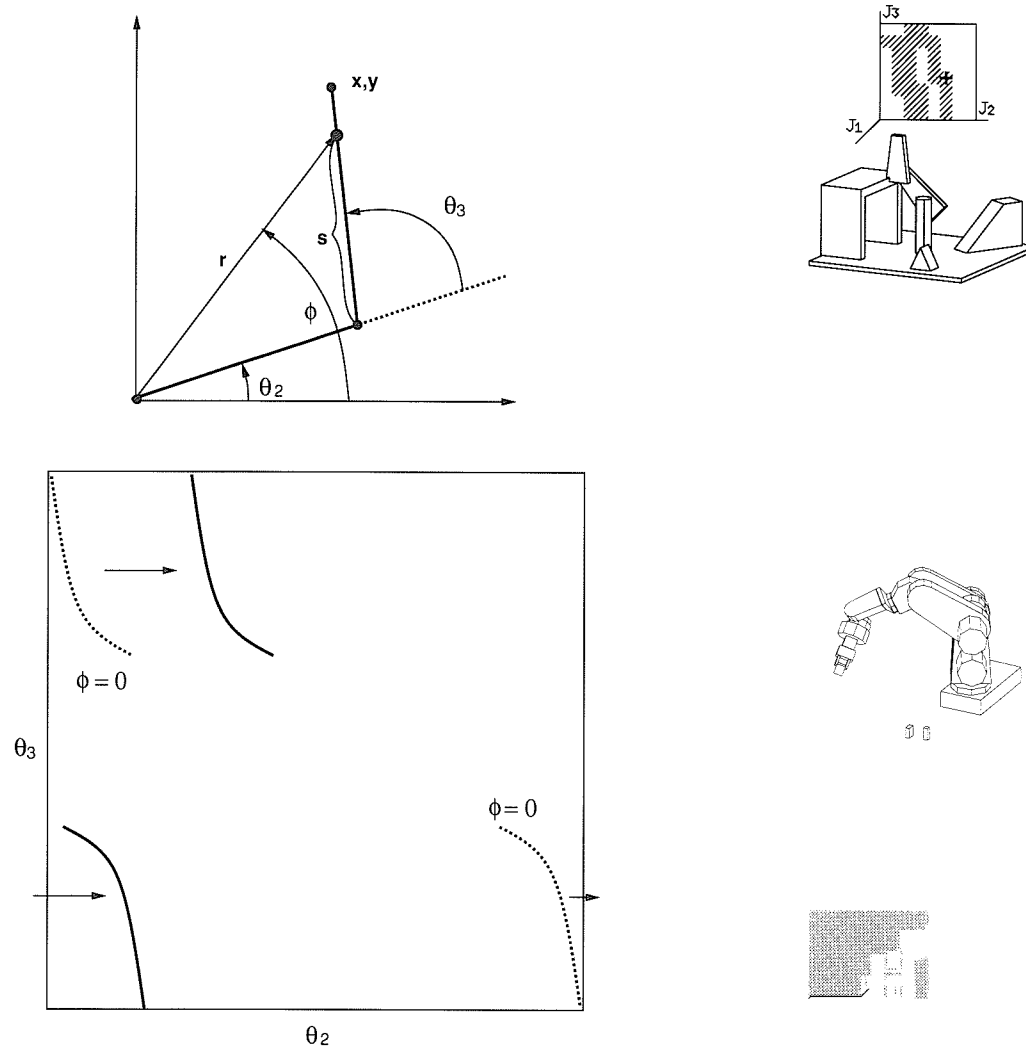
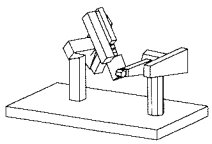
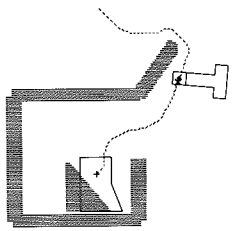


Figure 4.13
Top: A two-link planar manipulator. θ_2 and θ_3 are the joint variables. A point obstacle is shown at polar coordinates (r, ϕ). Configurations which collide depend on the parameters s, r, and ϕ. ϕ only affects the configuration as an offset of θ_2. Bottom: The configuration space obstacle of a point on the x-axis (dotted). As the polar angle, ϕ, of the point increases, the configuration space obstacle shifts along θ_2, but does not change shape (solid).





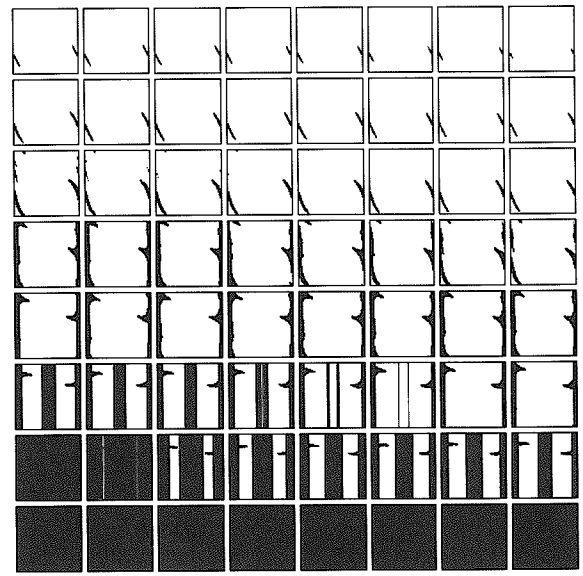


Figure 4.14
Primitive maps for some values of r for a planar robot. The link shapes for these maps are those of the second and third links of a Puma robot. The primitive maps are shown for $r = 0.0$ mm (at the lower left) to $r = 919.4$ mm (at the upper right) in increments of 14.6 mm. For small r, the primitive map is entirely blocked because the obstacle collides with link 1 in any orientation. Compare these maps with the C-space shown in Figure 4.1, page 59.

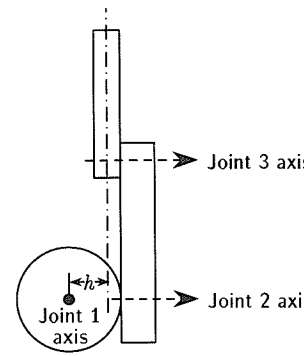
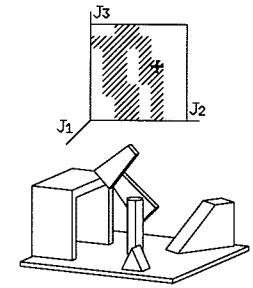


Figure 4.15
Top view of a Puma-like robot with a shoulder offset.

Figure 4.15 shows a top view of a Puma-like robot, including a shoulder offset, h. Let us, for the moment, neglect the actual shape and displacements of the arm links and assume that both links 2 and 3 reside in a plane. Then, given a point obstacle at x, y, z (with $\sqrt{x^2+y^2} \geq h$), there are two values of θ_1 for which that point will lie in the plane of the arm. Let $r' = \sqrt{x^2+y^2-h^2}$ and

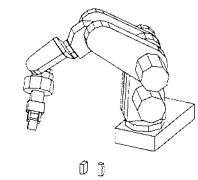


$$\psi = \text{atan}(y,x) + \text{atan}(\pm r', h) - \frac{\pi}{2} \tag{4.5.13}$$

$$\phi = \text{atan}(z, \pm r') \tag{4.5.14}$$

$$r = \sqrt{r'^2 + z^2} \tag{4.5.15}$$

Note that there are two sets of r, ϕ, ψ , one for each of the two roots for r'. These correspond to the two values of θ_1 which cause the plane of the arm links to intersect the point obstacle. Call the values of ψ and ϕ resulting from the positive root ψ_1 and ϕ_1; call the values resulting from the negative root ψ_2 and ϕ_2. We can then write the parametric equations describing the two configuration space loci for which there is collision between link 3 and the obstacle point as:



$$\theta_3 = f_3(r,s) \tag{4.5.16}$$

$$\theta_2 = \phi_1 + f_2(r,s) \tag{4.5.17}$$

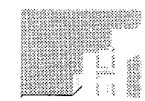
$$\theta_1 = \psi_1 \tag{4.5.18}$$

and

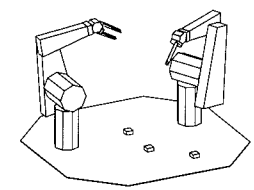
$$\theta_3 = f_3(r,s) \tag{4.5.19}$$

$$\theta_2 = \phi_2 + f_2(r,s) \tag{4.5.20}$$

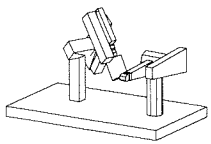
$$\theta_1 = \psi_2 \tag{4.5.21}$$



where f_2 and f_3 are as before. As in the planar case, the primitive configuration space obstacles form a one-parameter family of curves. In fact, it is exactly the same family, since the effect of θ_1 is merely to pick a plane within which the obstacle interaction happens.

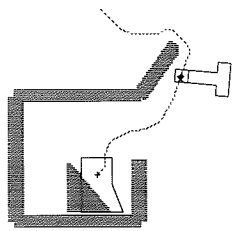


These equations ignore the shape of the arm out of the plane. For the Puma, the links are a constant width w perpendicular to their plane of motion. In that case, there are two *ranges* of ψ values over which the effects of a point obstacle are felt. The range can be approximated by treating the obstacle point as a sphere of diameter (at least) w. If the shape of the link out of the plane of motion were not a constant



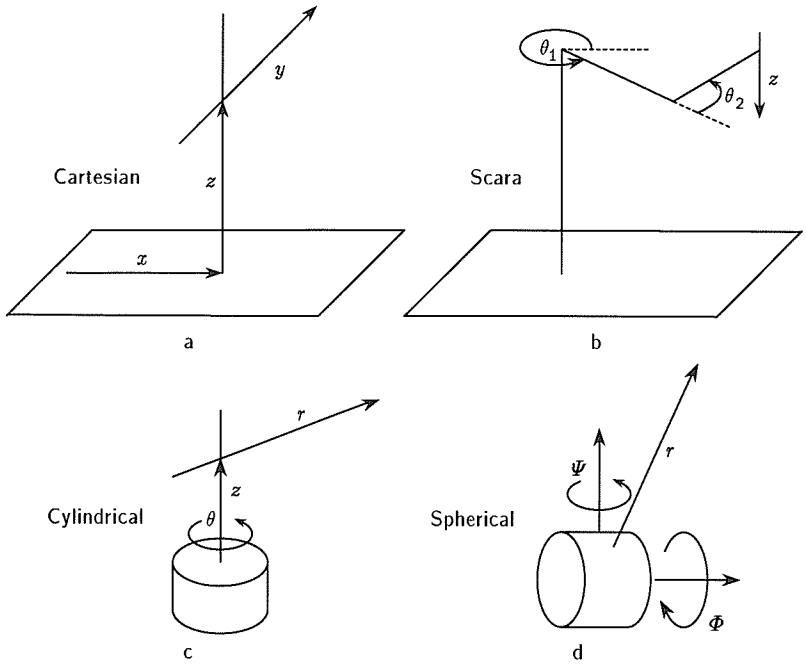
width, then we would need to use three-dimensional primitive C-space obstacles. To see this, consider attaching a spherical protrusion to the side of a planar link. Assume an obstacle point on the x-axis. Then the shape of the link in the plane containing the obstacle point is either the shape of the planar link or a circle, for those values of θ_1 which cause the protrusion to intersect the obstacle. Of course, this has drastic effects on the primitive obstacles, requiring the use of three-dimensional primitive maps so as to capture the variations as a function of ψ.

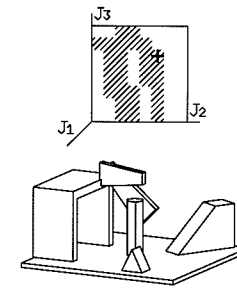
Other robot kinematics The approach described in the preceding sections is not limited to Puma-like robots. Whenever one has intersecting joint axes, one obtains the symmetries that these algorithms exploit. To illustrate this point, we will examine a few alternative kinematics and briefly point out how they can be handled (refer to Figure 4.16).



- Cartesian kinematics: There is a *single* three-dimensional primitive obstacle map for this kinematics. This represents the x, y, z positions of the robot arm that collide with a point at $(0, 0, 0)$. All other C-space obstacles are obtained by superimposing shifted copies of this single basic obstacle shape.
- Scara kinematics: If one focuses only on the vertical bar and gripper, ignoring the first two links, then this is identical to Cartesian kinematics. One can take into account the first two revolute links by computing the θ_2, θ_3 obstacles for the links and mapping each θ_2, θ_3 obstacle points into the corresponding forbidden x, y position of the third link.
- Cylindrical kinematics: If one treats the last link as planar with fixed width, as we did for the Puma, then the primitive obstacles for this kinematics form a one-parameter family of two-dimensional obstacles indexed by r. Shifts in the ψ and z directions are used to construct the final configuration space obstacles. For arbitrary link shape, we need the primitive obstacles to be three-dimensional.
- Spherical kinematics: The situation is more complicated; we actually need a two-parameter family of primitive maps. We will consider this case in more detail in Section 4.5.2.

Other primitive obstacles Our discussion has focused on the use of primitive configuration space obstacle maps due to point obstacles. Although this is conceptually the simplest choice, it is by no means the only or the most practical choice. We will see below that considerations of





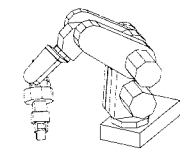
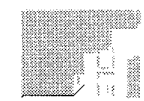
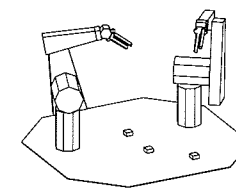


Figure 4.16
Other typical robot kinematics.

quantization suggest the use of circular or spherical obstacles. Branicky and Newman [5] have explored the use of lines and planes as primitive obstacles. At the expense of increasing the number of required primitive configuration space maps one can also use other primitives. For example, we could use square boxes at different orientations, but that would require a primitive obstacle map for each size and orientation of the box as well as for each radial distance of the box center from the robot base.

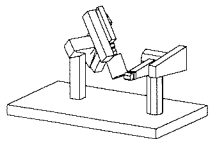


4.5.2 Grippers and wrists

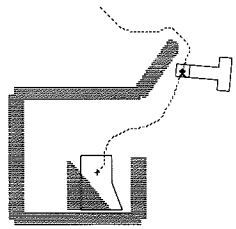


We have limited our attention thus far to the first three links of a robot. In this section we extend our ideas to full six-degree-of-freedom robots. We will show that for "wrist-decoupled" robots the applicability of families of primitive obstacle maps extends to the grippers. We

will also show how a simple search strategy can make use of fast three degree-of-freedom configuration space computation to implement a full six-degree-of-freedom path search.

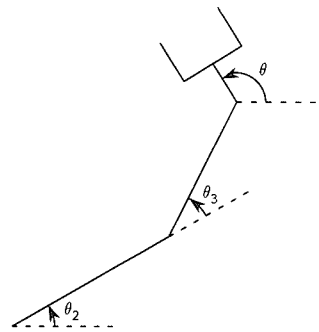


In the majority of non-redundant linked robots, the first three links serve to position the robot's **wrist**, a point where the last two or three joint axes intersect. It is well known that this type of wrist structure ensures the existence of a closed-form solution to the robot's kinematics. For related reasons, this design also ensures that the computation of the configuration space obstacles for the robot's gripper can be effectively decoupled from the computation of the obstacles for the arm. Therefore, we will assume that we are dealing only with "wrist-decoupled" robots. We can exploit the same kind of rotational symmetry that we have used to construct the configuration space around the base of the robot to characterize the obstacles for the robot gripper (and its payload). Furthermore, this approach is applicable to wrist-decoupled robots independent of the kinematic arrangement of the first three links.



The difficulty of dealing with the gripper is that the symmetry exists about the position of the "wrist," that is, the tip of the arm and not its base. Therefore, we have to perform a computation at each wrist position. This should not be surprising; it simply recognizes the fact that the gripper has six degrees-of-freedom. Fortunately, the computation is the same at each wrist position. Only the obstacles found in the neighborhood of each wrist position differ. This type of computation is, in principle, ideal for SIMD computers such as the Connection Machine. In practice, memory limitations have forced us to pursue a somewhat different approach for the six-dimensional case (see Section 4.5.3). We will illustrate our approach in the planar case, where it is easier to visualize.

Planar grippers Consider a simple planar robot gripper, such as the one in Figure 4.17. We are interested in determining the legal combinations of x, y, θ values for the gripper. That is, we want to construct a three-dimensional configuration space. Note that we are using values of θ, the orientation of the gripper about the global x axis, and not the values of the last joint angle. This choice is essential to enable us to consider collisions of the gripper independently of collisions of the arm.



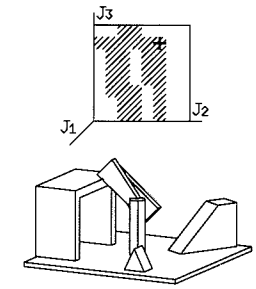
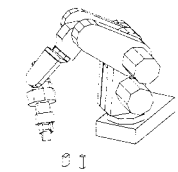


Figure 4.17
A two-link robot with gripper.

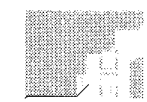
Our strategy is to characterize the forbidden positions and orientations of the "floating" gripper independent of any arm constraints. After this is done, we add the arm constraints, which amount to specifying that some positions and orientations of the gripper are not feasible because of arm collisions or limitations in the legal ranges of joint angles.



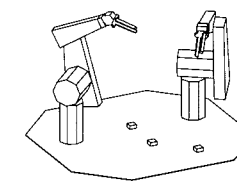
There are two basic approaches to computing the x, y, θ configuration space maps for the floating gripper:

1. Find all of the forbidden x, y positions of the wrist for each sampled θ value of the gripper.
2. Find all of the forbidden θ values for each x, y position of the gripper.

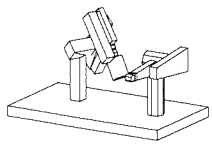
The different approaches lead to different algorithms that may be more or less suitable for parallel versus serial implementation.



For each θ, we can easily construct a C-space map representing all the x, y positions of the gripper for which there is a collision with an obstacle at position $(0, 0)$. For a point obstacle, this simply looks like a copy of the gripper rotated by $\pi + \theta$. The C-space obstacles due to more complex obstacles are obtained by combining shifted copies of this map. Essentially, there is a single three-dimensional primitive obstacle map for the floating gripper, representing forbidden values of x, y, θ due to a point obstacle at position $(0, 0)$.

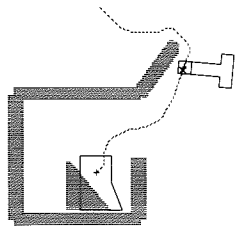


In our implementation, we have used the second of the listed approaches: characterizing the range of wrist angles for a given wrist



position. This process is analogous to that of building the C-space obstacles for the planar revolute manipulator. Earlier, we noted that rotations of the base coordinate frame of the revolute manipulator did not affect the shape of the C-space obstacles, but only their position in the C-space. In the case of a floating planar gripper, neither translations of the wrist nor rotations about the wrist affect the shape of the C-space obstacles. Only the distance between the point obstacle and the wrist matters. Therefore, the primitive obstacles for the gripper (at a particular wrist position) form a one-parameter family of *one*-dimensional C-space obstacles, representing forbidden values of θ and indexed by distance from the wrist position.

Three-dimensional grippers The three-dimensional case is analogous to the planar case. The difference is that now we have to deal with three wrist angles and three displacements. We examine the case of a gripper mounted on a spherical wrist; the extension of these remarks to other types of wrist construction is straightforward.

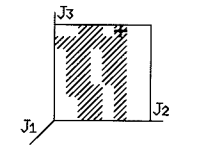


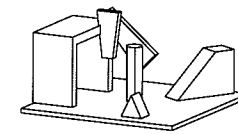
We will use the angles α, β, γ to represent the orientation of the gripper relative to the global Cartesian frame. The wrist angle α corresponds to the angle ψ in the spherical coordinate representation. The wrist angle β corresponds to the spherical angle ϕ. The wrist angle γ represents rotation about the axis defined by α and β. Given the $\theta_1, \theta_2, \theta_3$ joint angles of the arm that determine a wrist position, these wrist angles can easily be converted to joint angles for the wrist.

One crucial issue in handling a three-dimensional gripper is that the Euler-type angle specification used for the gripper only has one angle whose origin can be specified arbitrarily, namely α. Therefore, we require a two-parameter family of three-dimensional primitive maps, parameterized by values of β and r.

It is easy to see that the geometry of the system is unchanged when the base coordinate system is rotated by an arbitrary angle. Just as with the first link of the two-link robot, it follows that the shape of the primitive obstacles is independent of α. However, once the axis of rotation for α is chosen, a natural origin for β is defined; the obstacles are not independent of β. The three-dimensional C-space obstacle corresponding to a point changes in its θ_4 extent as the β coordinate of the gripper in contact with the point changes. As β approaches the axis of rotation for α, the size of an obstacle increases in the θ_4-dimension.

Compared to the primitive obstacle maps for the arm, the maps for the gripper include an extra dimension of the map and an extra parameter describing the family of maps. Although this would appear to increase the complexity of the problem, it will be seen that the simplicity of the algorithm to compute the C-space will not be affected. The fundamentals remain identical to those for the C-space of the arm.

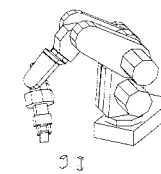


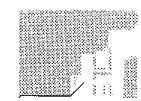


Although the three-dimensional gripper can be handled by simple algorithms, it still requires substantial amounts of time and space to construct the underlying six-dimensional configuration space. A useful approximation that reduces the dimensionality of the problem (and the required computation time and space) is to treat the gripper as being rotationally symmetric about the last joint axis. In that case, we can neglect the last wrist angle and operate only on two dimensional bitmaps, encoding the values of α and β.

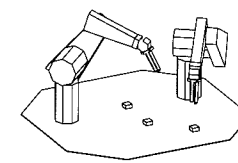
4.5.3 On-demand computation of obstacle maps

In general, one has to be wary of computing high-dimensional obstacle maps, as they require memory and computation that is exponential in the degrees-of-freedom. A single full six-degree-of-freedom configuration space bitmap with a sampling resolution of 11 degrees per index (such as are described in Section 4.5.4) would consume 128 megabytes of storage. Even our Connection Machine, as currently configured, contains only half that amount of memory. Our approach has been to compute obstacle maps only as needed to find a path. In particular, we construct the complete obstacle map for the three degrees of freedom of the arm and compute the maps for the wrist as necessary. We compute the wrist maps for a single x, y, z position of the wrist at a time, only requiring a three-degree-of-freedom map for the orientation of the wrist. The three-degree-of-freedom maps consume a more manageable 4096 bytes each.





Any safe path for the robot must lie in the free space of the configuration space of the arm minus the gripper. We can thus use that configuration space to direct a sequential search for a full six-degree-of-freedom path for the arm plus gripper. The search generates a path for the arm within the configuration space of the arm, and for each step of the arm's path, a motion of the wrist is computed which maintains safety from collision due to the gripper. This is easily accomplished by comparing the gripper configuration spaces of neighboring configurations along the



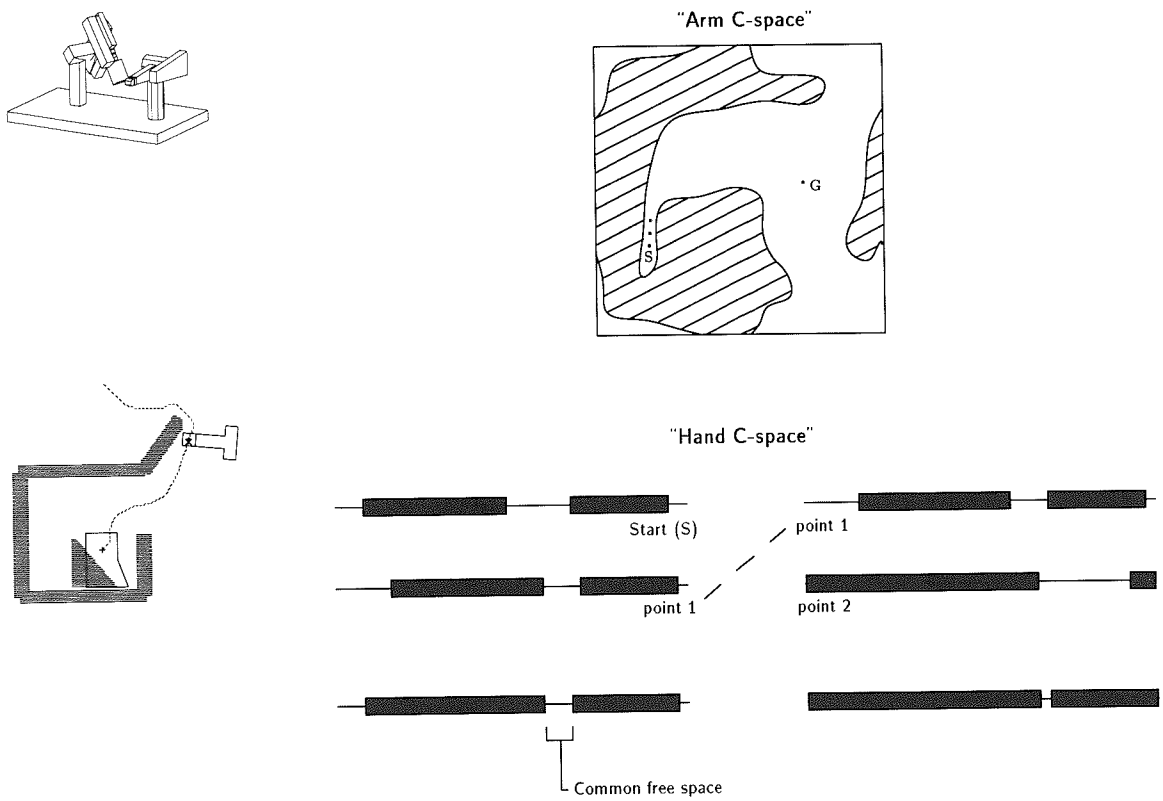
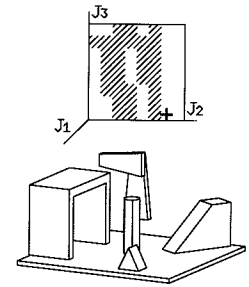


Figure 4.18
Finding a path using one arm configuration space and several gripper configuration spaces. For each possible path point in the arm C-space, we construct a gripper C-space. To move between neighboring points in the arm C-space, the gripper must avoid collisions in *both* gripper C-spaces. Gripper configurations in the common free space of the two gripper C-spaces are safe for that arm motion.

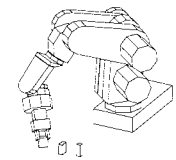
path to identify common free space (see Figure 4.18). With the gripper in any configuration in the common free space, the arm motion for that path step will be safe. All that remains is to find a path for the gripper into the common free space.



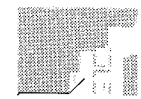
The parallel implementation of our approach (see Section 4.5.6) is sufficiently fast in computing configuration spaces that we can compute the gripper obstacle maps on demand as the search proceeds. As the search strategy tries each new step along the arm path, it computes the gripper map for the new configuration to compare it with the map for the current configuration. Whenever there is a common free space between the maps and there is a path for the gripper into the common free space, the search can consider that new step as a possibility. If no common free space or path can be found, then that branch of the search reaches a dead-end.

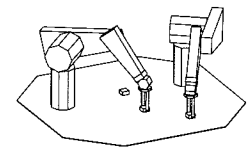
4.5.4 Serial algorithms

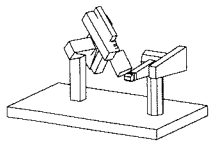
These observations about how configuration space maps can be constructed by superposition of shifted primitive maps lead us to straightforward algorithms for computing quantized configuration space maps for manipulators. We will explore the basic methods by presenting some serial algorithms for a Puma-like robot. The use of a Puma-like robot is merely illustrative, as we pointed out earlier, the algorithms apply to other kinematics with little change.



Bitmaps All of the algorithms presented in this paper operate on two or three-dimensional **bitmaps**. These bitmaps are used to represent Cartesian coordinate spaces, joint coordinate spaces, polar, and spherical coordinate spaces. In all cases, an entry in a bitmap represents a rectangular region of the corresponding space centered at the intersection of the grid lines. We call these rectangular regions the entry's **domain**. For example, if the quantization intervals for a two-dimensional Cartesian space are d_x and d_y, the i, j entry in the bitmap represents the region where $(i-0.5)*d_x \leq x \leq (i+0.5)*d_x$ and $(j-0.5)*d_y \leq y \leq (j+0.5)*d_y$. Therefore, the bitmap index corresponding to any value can be obtained by rounding the quotient of the value and the size of the quantization interval, i.e., round(x/d_x). (Note that the entry's domain includes both the upper and lower bound.)



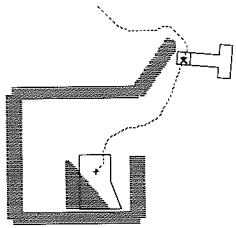




We present algorithms that treat the bitmap entries as representing points, not rectangular regions. We do this for simplicity of exposition. The extensions to the algorithms required to incorporate the more conservative interpretation are important, but relatively minor.

The algorithms we discuss make heavy use of a commonly used operation on bitmaps: that of writing a bitmap at a specified location in another bitmap, using an arbitrary boolean function to combine the existing bit values with the new values. This is a low-level primitive provided by all bit-mapped display systems and, in many cases is supported by hardware graphics accelerators. As a result, this operation is extremely fast. We will primarily use the inclusive-or operation to combine values and so we will write this operation as follows:

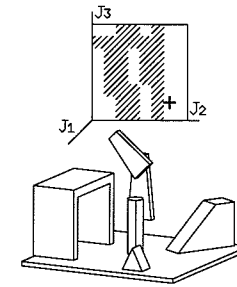
bitmap-ior(origin-bitmap, target-bitmap, shift-x, shift-y)



We first illustrate the algorithms for the two-link planar manipulator. The algorithms in this section make use of the following data structures:

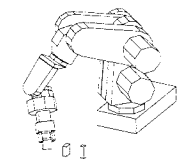
- $O(i,j)$ — an $n \times n$ bitmap representing the Cartesian obstacles. A bit is on in this bitmap if there is any part of an obstacle overlaps the bit's domain.
- $r(i,j)$ — an $n \times n$ array that stores the r index corresponding to the given Cartesian indices. This array is constant and only changes when the quantization increments change.
- $\phi(i,j)$ — an $n \times n$ array that stores the ϕ index corresponding to the given Cartesian indices. This array is constant and only changes when the quantization increments change.
- $P[r_{ij}](p,q)$ — a vector of primitive obstacle bitmaps, each representing the forbidden region in configuration space due to a point obstacle with $\phi = 0$. There is one bitmap for each quantized value of r, the radial distance of the point obstacle from the origin, for sampled values of $0 \leq r \leq l_2 + l_3$. The p and q indices correspond to quantized values of θ_2 and θ_3 respectively. This data structure needs to be computed only once for the robot.
- $C(p,q)$ — the computed C-space bitmap for all the obstacles. The p and q indices correspond to quantized values of θ_2 and θ_3, respectively. This array is first initialized with the robot's joint-angle limits before being used.

Serial algorithms for the two-link robot The simplest serial algorithm for building configuration space simply loops over the entries in the Cartesian obstacle bitmap and, for each filled entry, writes the primitive configuration space bitmap for that entry's r value into the global configuration space bitmap, at a θ_2 offset equal to that entry's ϕ value. The algorithm can be written as follows:

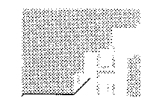


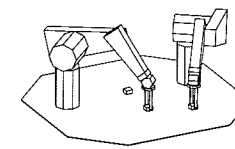
procedure two-link-from-xy-bitmap-serial
begin
  for i **from** 0 **to** n **do**
    for j **from** 0 **to** n **do**
      if $O(i,j) = 1$ **then bitmap-ior**$(P[r(i,j)], C, \phi(i,j), 0)$
end

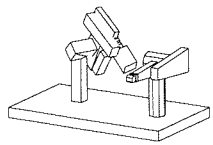
Note that primitive bitmaps and the global configuration space bitmap are of the same size (they both are uniformly sampled versions of S^2). Uniform sampling is essential to allow direct combination of differently shifted bitmaps. When we write the obstacle bitmap into the global map shifted by a specified value of θ_2 what we intend is that the obstacle bitmap be "wrapped-around" so that bits that are shifted off the 2π-edge reappear at the 0-edge. In practice, this can be implemented by making the global map span the range $0 \leq \theta_2 \leq 4\pi$. After all the primitive maps have been written into the global map, the section of the global map spanning $2\pi \leq \theta_2 \leq 4\pi$ can be combined (using the bitmap-ior operation) with the section spanning $0 \leq \theta_2 \leq 2\pi$.



The running time of this algorithm depends linearly on the number of bitmap-ior operations that it must perform. The number of bitmap-ior's required to compute a configuration space map using this algorithm grows with the *area* of the x, y obstacles. If we are searching for paths between robot configurations known to be safe, then we only need to characterize the boundary of the configuration space obstacles. Therefore, we can improve the algorithm above by only performing the bitmap-ior for Cartesian points that are on the boundary of a Cartesian obstacle, that is, not completely surrounded by other points. With this variation, the number of bitmap-ior's is proportional to the *perimeter* of the Cartesian obstacles rather than to their area. This can be a substantial saving in problems involving large obstacles.



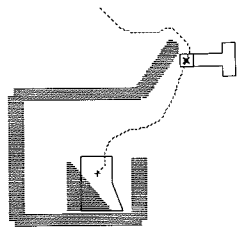




The third dimension The algorithm for constructing the $\theta_1, \theta_2, \theta_3$ configuration space is essentially the same as the two-degree-of-freedom algorithm. The key differences are that the mapping from x, y, z to r, ϕ, ψ is not unique and that the resulting configuration space is three-dimensional.[3] This imposes some changes in the required data structures:

- $\psi_1(i,j,k), \psi_2(i,j,k)$ are the indices corresponding to the two solutions for ψ given in (4.5.13).
- $\phi_1(i,j,k), \phi_2(i,j,k)$ are the indices corresponding to the two solutions for ϕ given in (4.5.14).
- $C[o](p,q)$ is a vector of θ_2, θ_3 bitmaps. The o-index corresponds to a quantized value of θ_1.

The other data structures are simply replaced by their direct three-dimensional counterparts. The algorithm for a three-link Puma-like robot can then be written as:

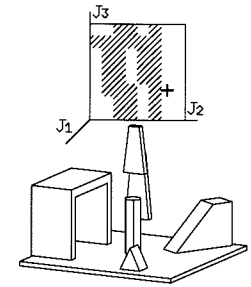


procedure three-link-from-xy-bitmap-serial
begin
  for i **from** 0 **to** n **do**
    for j **from** 0 **to** n **do**
      for k **from** 0 **to** n **do**
        if $O(i,j,k) = 1$ **then begin**
          bitmap-ior$(P[r(i,j,k)], C[\psi_1(i,j,k)], \phi_1(i,j,k), 0)$
          bitmap-ior$(P[r(i,j,k)], C[\psi_2(i,j,k)], \phi_2(i,j,k), 0)$
        end
end

Note that the primitive configuration space maps, $P[r]$, are still two-dimensional. This property follows from our assumption that the arm links reside in a plane. For more general link shapes and kinematics, the primitive maps will be three-dimensional, requiring a three-dimensional bitmap3-ior operation. Such an operation can be implemented easily in terms of the two-dimensional operation, in the same way that the two-dimensional operation is built out of the fundamental one-dimensional operations provided by traditional computer instructions.

[3] We will see in a later section that we can get around this problem by beginning in the r, ϕ, ψ -space and avoiding Cartesian space altogether.

4.5.5 Planar grippers

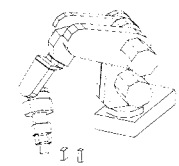


We are interested in determining the legal combinations of x, y, θ values for the gripper. Recall that we are using values of θ, the orientation of the gripper about the global x axis, and not the values of the third joint angle. As we discussed earlier, there are two ways to approach the computation:

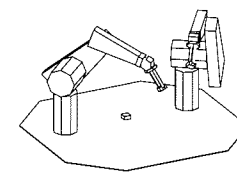
- for each reachable x, y wrist position, find the legal range of θ, or
- for each value of θ, find the legal range of x, y wrist positions.

We will consider both approaches here.

For each x, y We can characterize the point obstacles by their polar coordinates relative to a coordinate frame which is aligned with the base frame of the robot, but with its origin being the wrist position. Call these relative coordinates r^*, ϕ^*. The value of r^* for the obstacle point determines the forbidden ranges of θ (this is analogous to the primitive θ_2, θ_3 obstacle maps for the manipulator) and the value of ϕ^* determines the offset in θ (as it does for the manipulator). Note that only bitmap entries whose r^* value is less than or equal to the gripper length l_h need to be considered. Letting n_h be the bitmap index corresponding to l_h we can construct the θ obstacles at every i, j position of the wrist as follows:

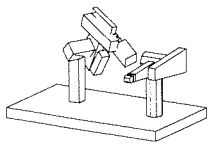


```
procedure planar-gripper-from-xy-bitmap-serial-1
begin
  for i from -n_h to n + n_h do
    for j from -n_h to n + n_h do
      if free(i, j) then
        begin
          for i* from -n_h to n_h do
            for j* from -n_h to n_h do
              if O(i + i*, j + j*) = 1 then
                bitmap-ior(P[r*(i*, j*)], C(i, j), φ*(i*, j*))
        end
end
```



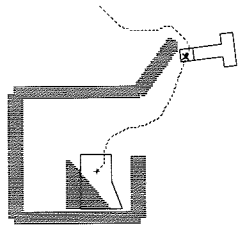
The inner loop is essentially identical to the two-link planar algorithm. The key difference is that the primitive bitmaps $P[r]$ and the contents of each $C(i, j)$ are one-dimensional, representing ranges of θ. The test

$free(i, j)$ is meant to test whether the wrist position indexed by i, j is reachable by the arm without collisions; this can be tested using the configuration space map for the arm.



For each θ For each θ we can easily construct a bitmap representing the x, y positions of the gripper for which there is a collision with an obstacle at $(0, 0)$. For a point obstacle, this simply looks like a copy of the gripper rotated by $\pi + \theta$.

We can represent the three-dimensional configuration space bitmaps required in this problem as vectors of two-dimensional bitmaps. In the algorithm below, p loops over quantized values of θ. Therefore, $C[p]$ denotes the two-dimensional x, y bitmap for the particular value of θ indexed by p. The complete algorithm is simply:

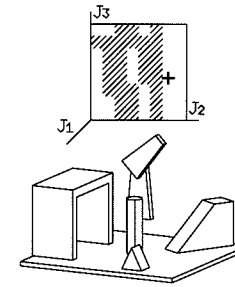


procedure planar-gripper-from-xy-bitmap-serial-2
begin
  for p **from** 0 **to** m **do**
    for i **from** 0 **to** n **do**
      for j **from** 0 **to** n **do**
        if $O(i, j) = 1$ **then bitmap-ior**$(P[p], C[p], i, j)$
end

The number of bitmap-ior operations required to execute this algorithm is equal to the area of the obstacles times the number of θ samples. We can reduce this number in several ways:

- We can limit the computation to points on the boundary of the obstacles, that is, to filled points not completely surrounded by other filled points. This reduces the number of bitmap-ior operations to the perimeter of the obstacles times the number of θ samples. This restriction produces an incomplete configuration space bitmap, but one that can be used to plan motions as long as we independently check that the start and goal points are collision-free.
- If we have the input obstacles represented as a **quad-tree**, we can apply the basic algorithm to the "boxes" at the leaves of the quad-tree. In fact, since the boxes at the leaves of the tree have dimensions which are powers of 2, we can precompute the primitive obstacle bitmaps for each of these dimensions. Then, the time to construct a configuration space map depends only on the number of leaves in the quad-tree times the number of θ samples.

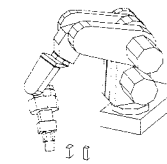
4.5.6 Parallel algorithms



The parallel version of our approach is both simpler and trickier than the serial version. It is simpler because the parallelism unrolls all (or nearly all) the loops from the serial version. It is trickier because the organization of the data in the SIMD computer must be carefully laid out to take full advantage of the parallelism offered. Our algorithm runs in time that depends only on the resolution desired. It is independent of the number of obstacles or their size.

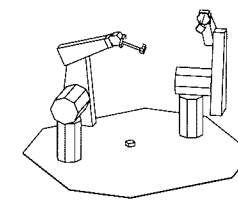
The fundamental algorithm is unchanged whether we're generating the C-space map for the two-link planar robot, the three-dimensional Puma-like robot or the general 3-D gripper. The layout of the data in the Connection Machine is the only difference, described below.

The Connection Machine Our Connection Machine [28] has 8192 physical processors which can be configured on demand to simulate a conceptually unbounded number of virtual processors arranged in an arbitrarily dimensioned grid. This capability suggests several natural mappings of parameters of the path planning problem to axes of the virtual Connection Machine. Unfortunately, large numbers of virtual processors (large ratios of virtual-to-physical processors) make unreasonable demands on the available physical memory of the machine, and also increase program running time approximately linearly with the virtual processor ratio. As a result, our actual implementation trades off natural implementation strategy against memory and time.



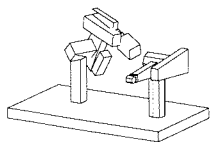
Besides the bitmap-ior operation, the Connection Machine offers two other operations which our parallel algorithms exploit. The first operation is nearest-neighbor communication between processors. Our algorithms use this communication to shift the primitive obstacle bitmaps. The second operation is a **scan** operation, which combines data from all processors along a specified dimension of the processor grid using an associative arithmetic function, such as logical-or. This scan operation runs in essentially constant time.

Data representation As in the serial case, we first illustrate the algorithms for the two-link robot. Our parallel algorithms make use of the data structures described below.

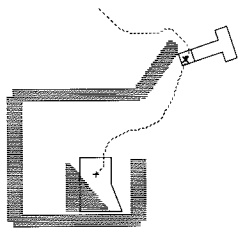


We need to represent three data structures in the Connection Machine: the bitmap representing the Cartesian obstacles, the primitive obstacle

bitmaps, and the configuration space bitmap. It is natural to associate the primitive obstacle bitmaps and the configuration space bitmap with a two-dimensional processor grid, with each processor representing one bit. It is not immediately obvious how to represent the Cartesian obstacles to efficiently exploit the machine.



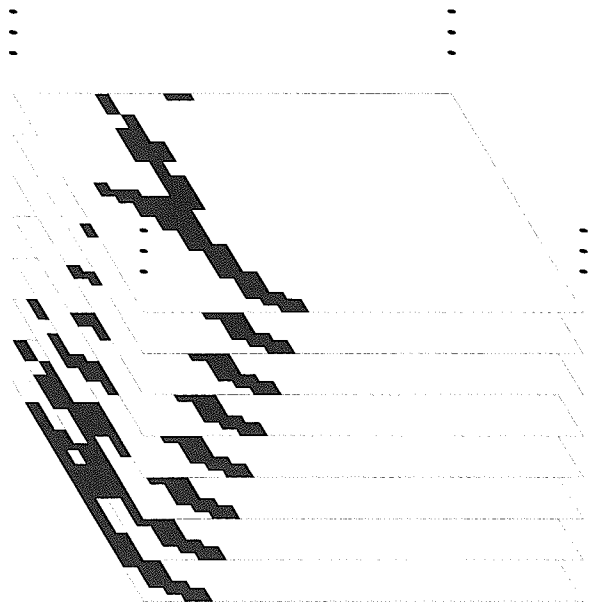
In the serial algorithm, we transformed each Cartesian obstacle bit into a (r, ϕ) coordinate pair which was used to index into the vector of primitive obstacle maps and to offset the primitive map before combining it into the configuration space map. We can imagine, instead, generating the initial obstacle bitmap directly in polar coordinates.[4] This polar-coordinate obstacle bitmap is denoted $O_p(r, \phi)$. Here, r and ϕ denote indices on the bitmap, and represent quantized values of the polar coordinates. The bit $O_p(r, \phi)$ will be on if there is an obstacle in Cartesian space intersecting the domain of that bit—if there is an obstacle at the point (r, ϕ).

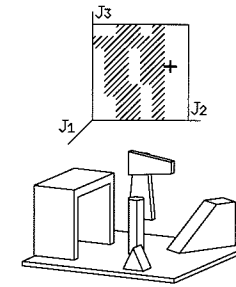


We configure the Connection Machine into a *four*-dimensional, $n \times n \times n \times n$ grid, indexed by p, q, r and ϕ. The reason for this will be seen in the procedure for computing the configuration space, which will consist of one scan along each of the dimensions of the Connection Machine. As in the serial algorithm, the p and q indices correspond to quantized values of θ_2 and θ_3. The r and ϕ indices will represent the (quantized) polar coordinates.

To see the layout of the data in the CM, consider a two-dimensional sub-grid indexed by a particular pair of (r, ϕ)—a p-q plane (see Figure 4.19). Within this sub-grid, we store the primitive obstacle map, $P[r]$, offset along the θ_2-axis (i.e., the p-axis) by the value of ϕ used to select this sub-grid. The effect is that we will store in a processor with indices (p, q, r, ϕ) the primitive obstacle bit $P[r](p - \phi, q)$. (The subtraction $p - \phi$ must, of course, be computed modulo n.) Call this the P-bit. Each primitive obstacle map, $P[r]$ will be represented in the Connection Machine n times, once for each value of ϕ. Since the primitive maps depend only on the robot and the resolution of the quantization, the maps need to be downloaded into the Connection Machine only once.

[4] In fact, using a non-local processor communication feature of the Connection Machine, the polar-coordinate obstacle bitmap can be computed from the rectangular-coordinate bitmap. It is easier, however, to directly compute the polar-coordinate bitmap: each processor tests whether the (polar-coordinate) domain it represents intersects any obstacle. In parallel, this bitmap can be computed linearly with the number of obstacles.





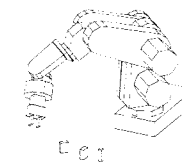
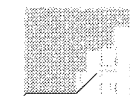


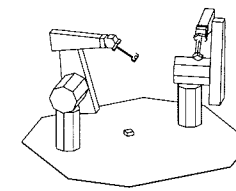
Figure 4.19
The primitive maps as stored in the Connection Machine. This stack illustrates the storage for a single value of r. Each plane represents a θ_2-θ_3 plane indexed by a particular value of ϕ. (For the resolution shown, there would be 32 of these planes.) The map is shifted according to the ϕ index of the processor. The r-dimension (not shown) controls the shape of the primitive configuration space obstacle.

The two-dimensional polar-coordinate obstacle bitmap, $O_p(r, \phi)$, is duplicated for all values of p and q; that is, each processor, (p, q, r, ϕ), stores the obstacle bit for coordinates (r, ϕ). For each r and ϕ the entire θ_2-θ_3 plane is either 1 or 0, depending on $O_p(r, \phi)$. Call this the O_p-bit.

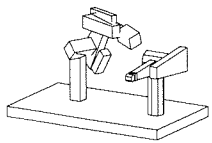
The final configuration space, $C(p, q)$, will be generated in the two-dimensional, p-q sub-grid selected by $(r = 0) \wedge (\phi = 0)$.



Parallel algorithm for the two-link robot Assuming that the obstacle bitmap has already been stored in the Connection Machine, the computation of the configuration space only needs to combine the primitive maps as selected by the obstacle bitmap. This is the same as in the serial algorithm. In the parallel version, however, the combination is done in just two scan operations.



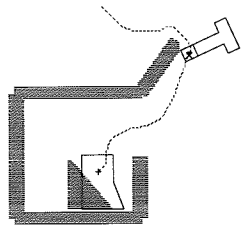
We look again at the serial procedure, reproduced here (modified to use the polar-coordinate obstacle map):



procedure two-link-from-polar-bitmap-serial
begin
  for r **from** 0 **to** n **do**
    for ϕ **from** 0 **to** n **do**
      if $O_p(r,\phi) = 1$ **then bitmap-ior**$(P[r], C, \phi, 0)$
end

We see that a bit in the configuration space, $C(p,q)$, is turned on if there is any r and ϕ such that $O_p(r,\phi) = 1$ and $P[r](p-\phi,q) = 1$. This condition is easily computed: compute the conjunction of the O_p-bit and the P-bit, and scan using logical-or along the r and ϕ dimensions. The result will be the configuration space.

This is the entire parallel algorithm:



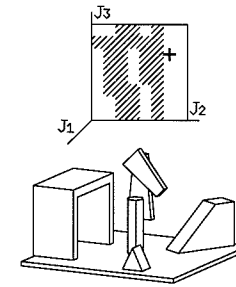
procedure two-link-from-$r\phi$-bitmap-parallel
begin
  $C \leftarrow$ **scan**(**logior**, $P \wedge O_p$, r-axis, **toward-zero**)
  $C \leftarrow$ **scan**(**logior**, C, ϕ-axis, **toward-zero**)
end

This algorithm runs in essentially constant time, since only the scan operations are used. The obstacle bitmap, O_p, can be computed directly in the Connection Machine in time that is linear in the number of obstacles. Contrast this with the serial algorithm which, at best, runs in time linear in the combined *perimeter* of all the obstacles.

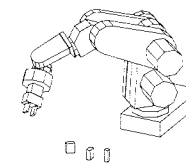
One way to visualize this algorithm is to imagine the two-dimensional $p \times q$ grid. At each grid point is stored a duplicate copy of the two-dimensional $O_p(r,\phi)$ bitmap. Also at each grid point are stored n^2 bits from the primitive obstacle bitmaps: $P[r](p-\phi,q)$, for all values of r and ϕ. The two scans then, in essence, select for each r and ϕ, the primitive maps and appropriate shifts indicated by $O_p(r,\phi)$. The processors at $(r=0) \wedge (\phi=0)$ will then contain the bits of the configuration space map, $C(p,q)$.

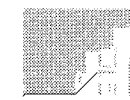
Another way to view the algorithm is as a projection from four dimensions to two of the primitive maps as selected by the obstacle bits.

The third dimension Conceptually, the extension of the two-link, two-dimensional parallel algorithm to one for a full three-dimensional, Puma-like robot is trivial. As mentioned above, the first joint rotation, θ_1, simply selects which plane the remaining two links operate in, and the algorithm operates within that plane identically to the two-dimensional case. Thus, we need simply add a single dimension to our processor grid and represent our original obstacle bitmap in a kind of spherical coordinate system indexed by (r, ϕ, ψ) triples. This spheric-coordinate obstacle map is denoted by $O_s(r, \phi, \psi)$. (These are not the usual spherical coordinates, since the shoulder offset of the Puma robot yields two values for ψ and ϕ for a single point. As with O_p, the O_s bitmap is easily computed directly, since the mapping $(r, \phi, \psi) \rightarrow (x, y, z)$ is many-to-one, and each (r, ϕ, ψ) domain can be easily tested for intersection with obstacles.) The algorithm is identical to the two-dimensional case, with the substitution of O_s for O_p.

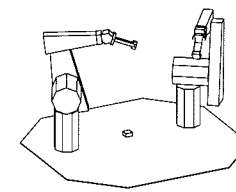


Unfortunately, on our Connection Machine, five dimensions each with 32 processors on each axis (a resolution of 11 degrees) requires 32 million virtual processors, or a virtual processor ratio of 4096 : 1. This ratio is totally unreasonable in terms of both execution time and memory availability. The algorithm is modified to alleviate this problem by combining the bits for each data structure along θ_3-dimension into bit-vectors within each processor. Also, instead of storing all the shifted copies of the primitive maps in the ϕ-dimension, we loop (serially) over all values of ϕ, using the nearest-neighbor communication facility of the Connection Machine to shift the primitive maps, combining the maps into the configuration space at each step, as indicated by the O_s bitmap. This modification of the algorithm still runs in time that depends only on the resolution; it increases linearly with the resolution.

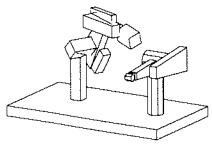


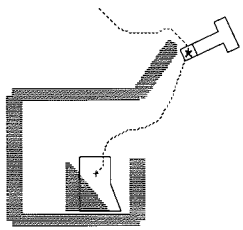


Parallel algorithm for the three-dimensional gripper The parallel algorithm for the three-dimensional gripper is as easy to construct as the three-dimensional arm was from the two-dimensional arm. This is because the algorithm is, in essence, identical. The primary difference is simply that, as discussed in Section 4.5.2, the primitive obstacle maps are three-dimensional, and they are indexed by two parameters, β and r, as well as being shifted by α. Conceptually, this leads to a six-dimensional processor grid, with axes θ_4, θ_5, θ_6, α, β, and r. In the implementation this is reduced to three dimensions by looping over α and compressing



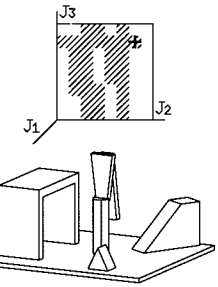
the θ_5 and θ_6 dimensions into a two-dimensional bitmap in each processor. Other than the size of the bitmap stored in each processor, the algorithm is identical to the three-dimensional arm algorithm. (For the arm, n bits are stored in a vector; for the gripper n^2 bits are stored in a two-dimensional bitmap.)





5 Grasp Planning

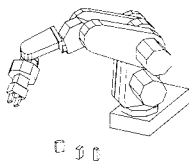
The HANDEY grasp planner chooses a grasp on an object and plans motions to approach and depart from this grasp. The grasp planner is invoked as follows:



Grasp(*goal*, *robot*, *world*, *depart*?)

If the *depart*? flag is true, the grasp planner will generate two plans. The first one is to approach and grasp the part specified in the goal at its pickup pose in the specified world. The second is to release the part at its putdown pose and to back away slightly from the grasp. If the *depart*? flag is false, only the approach plan is computed.

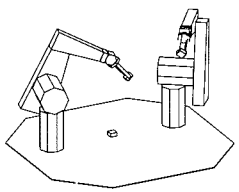
Recall the key steps in planning a pick-and-place operation, illustrated in Figure 2.6, page 23. The grasp planner computes an approach configuration, a grasp configuration, a departure configuration, and paths connecting the approach configuration to the grasp configuration and the grasp configuration to the departure configuration. These configurations need to be chosen so as to satisfy the interacting constraints on pick and place operations discussed in Chapter 2.



This chapter develops a method for simultaneously satisfying many, but not all, of the pick-and-place constraints. In particular, although the approach (and departure) configuration is chosen to be in a region reachable by the robot, the grasp planner cannot ensure that the gross motion planner will be able to find a path from the initial configuration to the approach configuration. Also, as we have seen, the choice of a grasp may preclude existence of a path from pickup to putdown for the robot holding the grasp object; this is currently ignored when picking a grasp. Neither of these limitations has proven to be a problem in practice.

5.1 Basic assumptions

As with all of HANDEY, the grasp planner operates on polyhedral models of grippers and parts. The grasp planner, however, makes a number of additional simplifying assumptions:



1. The robot grasps objects using a parallel jaw gripper.
2. A grasp must have the inner face of each gripper finger in contact with a **grasp feature** of the object. At least one such contact must be a planar face/face contact.

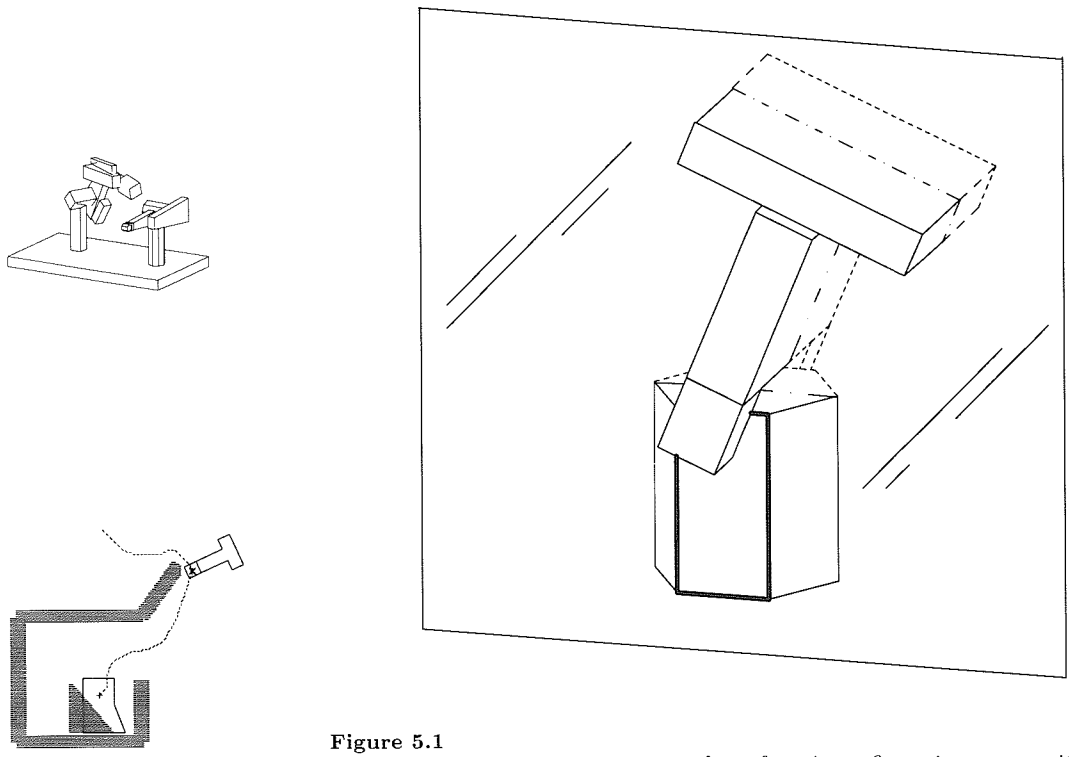


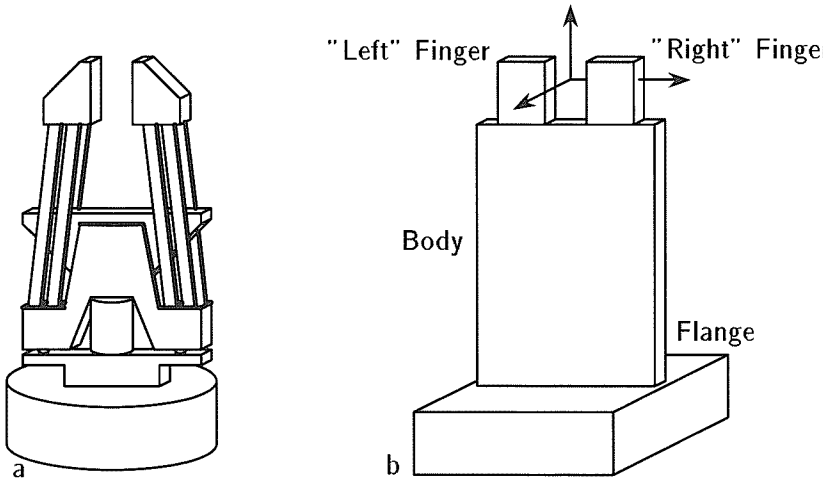
Figure 5.1
A grasp must have at least one inner face of a gripper finger in contact with a face of the object (highlighted). This face/face contact determines the grasp plane. Motion to and from the grasp is constrained to be parallel to this grasp plane.

3. The objects grasped are relatively light and their surfaces are sufficiently rough so that grasping stability can be easily guaranteed.
4. The object face involved in a grasp establishes a **grasp plane**. Approach to and departure from a grasp is done with the fingers moving parallel to this plane (see Figure 5.1).

We will address each of these assumptions in turn.

5.1.1 The gripper model

The parallel-jaw gripper used during most of HANDEY's development is shown in Figure 5.2(a). The grasp planner simplifies this to the model depicted in Figure 5.2(b). This figure also shows the **finger frame**,



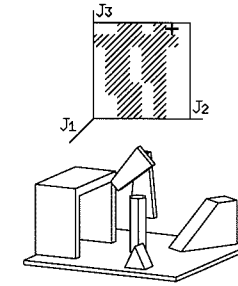
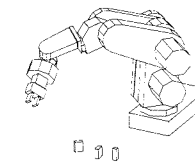
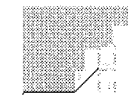


Figure 5.2
Parallel-jaw gripper models: (a) an accurate model of a gripper and (b) the simplified model used in the grasp planner.

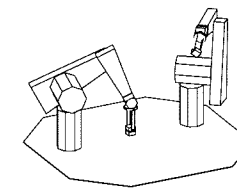


used in determining the position and orientation of the gripper. The origin of the finger frame is the **reference point** for the gripper. The key modeling assumption in the gripper representation is the division of the gripper into several brick-like components, whose projections into the grasp plane are rectangles aligned with the axes of the finger frame. The dimensions of these components are specified as part of the robot model.

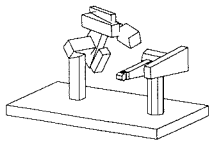
5.1.2 Grasps

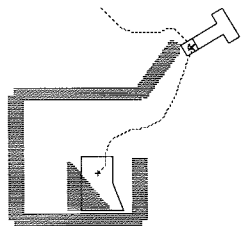


Grasps can be categorized by the type of the grasped features, that is, faces, edges, or vertices, with which the interior of the fingers, the **grip surfaces**, are in contact. The current version of HANDEY only considers grasping parallel faces whose projection along their common normal overlap. The method described in this chapter, however, is directly applicable to any grasp involving at least one face and another suitable feature. That is, possible grasps could include:



1. two parallel faces,
2. a face and a parallel edge, or
3. a face and a vertex.





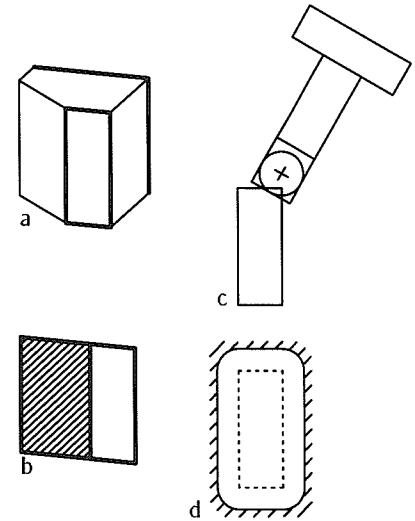


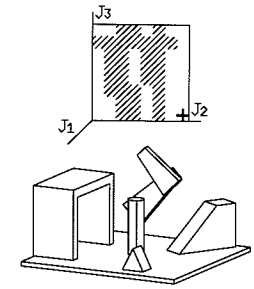
Figure 5.3
Overlap stability criterion. (a) Use front and back faces as grasp faces. (b) Only the intersection of these faces offers a viable grasp (non-hatched area). (c) Some point on gripper fingers must overlap graspable area. (d) Grasp criterion is met if gripper reference point is inside hatched boundary.

The pair of features must overlap in their projection in the grasp plane. The treatment of each of these grasps can be reduced to the parallel-face case simply by treating the edges and vertices involved in the grasp as small faces, parallel to the other grasped face.

5.1.3 Grasp stability

When choosing among candidate grasp faces and when choosing grasp poses on a given grasp face, one should consider stability: the resulting grasp must resist disturbance forces, both static—arising from the force of gravity, and dynamic—arising from the motion of the robot. The subject of stable grasping has received extensive attention in the published literature (see Section 2.5). We have chosen to neglect sophisticated stability considerations in HANDEY; we use a very simple geometric overlap criterion that is sufficient when dealing with light parts that are not slippery.

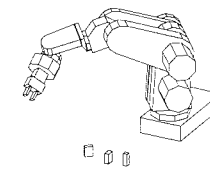
The grasping stability criterion used in HANDEY is as follows: we require that some point interior to a circle inscribed on the gripper fingers overlaps the intersection of the grasp faces (see Figure 5.3). Using an inscribed circle to define overlap has several practical advantages: it is insensitive to the orientation of the fingers and it is easy to modify the criterion to insure a minimum amount of overlap (smaller radii insure greater overlap).



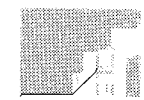
It is possible to extend this simple grasp criterion to include a more complete treatment of stability. In practice, all HANDEY needs is some way to identify a subset of the grasp plane which contains valid targets for grasps. Currently in HANDEY, the choice of this subset is based purely on geometric criteria, but it could be based on force and other stability criteria as well.

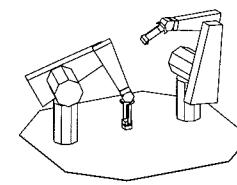
5.1.4 Planar motion

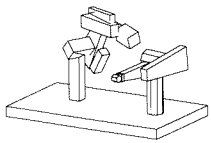
When the gripper is about to grasp an object, the finger grip surfaces must already be parallel to the grasp plane, and the only motions the gripper can make are those which are nearly parallel to the grasp plane. The grasp planner exploits this approximate constraint by planning motions in which the gripper is restricted to the grasp plane (see Figure 5.1). Thus, with little loss of generality the grasp planning problem can be reduced from six degrees-of-freedom to the three degrees-of-freedom required to specify the position, (x, y), and orientation, θ, of the gripper in the grasp plane. These three parameters define the **gripper pose**. The position and orentation of the gripper at the point it grasps the object is called the **grasp pose**.

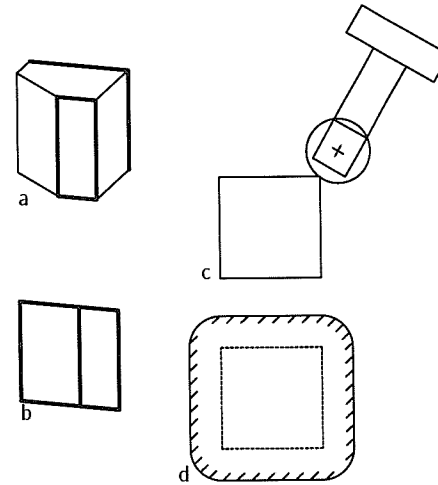


The gripper motion in the grasp plane will take place between the gripper's **approach pose** to the **grasp pose** and from there to the **departure pose**. At the gripper approach and departure poses the responsibility for planning motions will be transferred between the gross motion planner and the grasp planner. The gross motion planner operates at a lower resolution than the grasp planner, hence it should not be expected to plan motions that bring the robot or the grasp object too close to other objects. Therefore, the criterion for choosing the approach and departure gripper poses is (approximately) that the projection of the fingers do not overlap the grasp faces. This constraint is implemented in HANDEY by requiring that, at the approach and departure poses, the









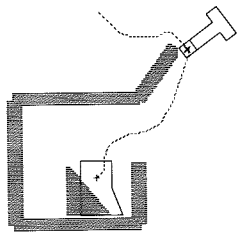


Figure 5.4
Approach/Departure criterion. (a) Grasp faces. (b) Union of grasp faces. (c) No point on gripper fingers may overlap the union of the grasp faces. (d) Non-grasp criterion is met if gripper reference point is outside hatched boundary.

union of the grasp faces does not overlap a circle circumscribed about the gripper fingers (see Figure 5.4).

5.2 Grasp planner overview

Briefly, the HANDEY grasp planner performs these computations:

1. Choose a pair of faces to grasp (thus establishing a grasp plane).
2. Project obstacles at the pickup and putdown locations into the grasp plane.
3. Characterize grasp, approach, and departure regions relative to the grasp object,
4. Compute approach and departure C-space slices as required.
5. Search the C-space for the following: grasp pose, approach pose, departure pose, approach path, and departure path.

This process is outlined in Figure 5.5. Each of the steps in turn is discussed more fully below.

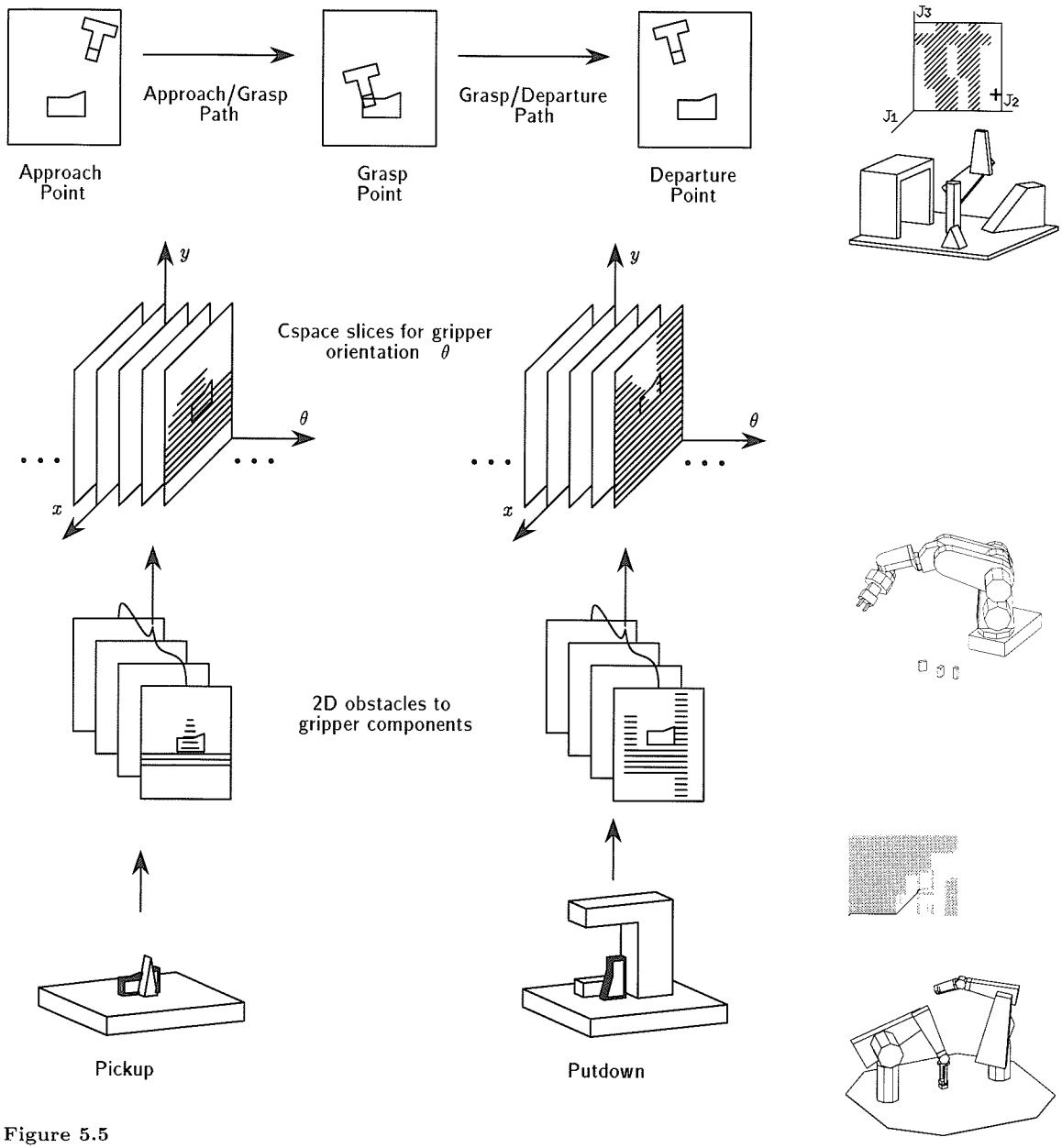
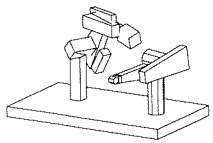


Figure 5.5
Conceptual outline of the grasp planner



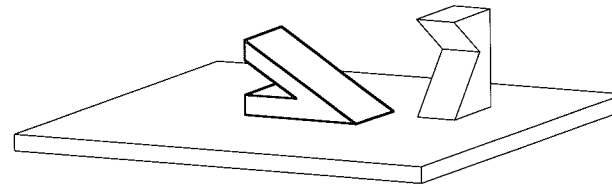
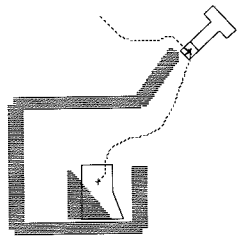


Figure 5.6
Grasp the 'V' shaped object



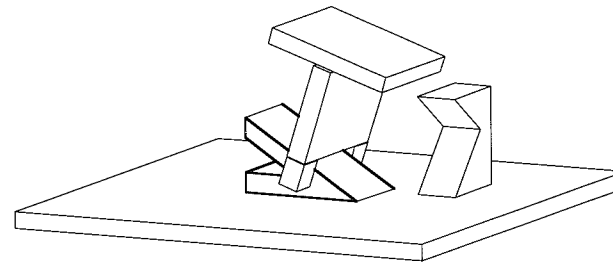
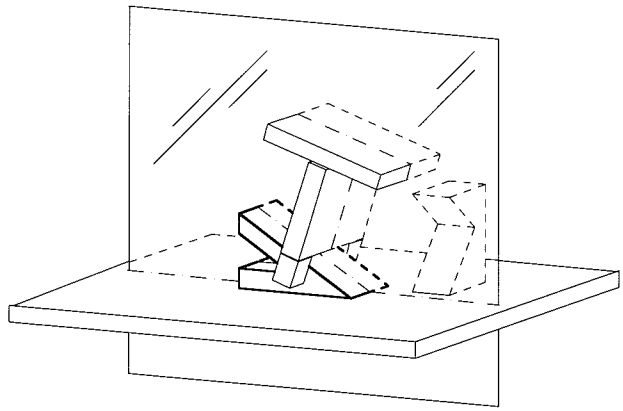


Figure 5.7
Choose the 'V' shaped face pair

5.2.1 Choosing grasp faces

Consider the example shown in Figure 5.6. Here a pick-and-place operation must be planned for the highlighted object. The first step is to choose a pair of grasp faces for investigation. Usually there will be multiple pairs of grasp faces available for grasping. The grasp planner attempts to rank these possibilities in order of decreasing likelihood of finding a viable grasp pose. It does this at low resolution before proceeding with a more detailed examination at higher resolution. Choosing among the possible grasps involves these steps:

1. Compile a list of all pairs of faces which lie on parallel planes and whose normals point away from each other.
2. For each of these pairs compute a measure of the area available for grasping considering obstacles near the pickup and putdown positions. (This will be explained more fully below.)



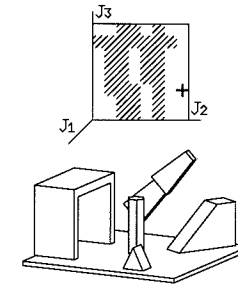
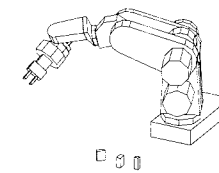


Figure 5.8
Grasp plane

3. Rank the face pairs in order of decreasing graspable area.
4. Select the pair with the largest grasp area for consideration at higher resolution.



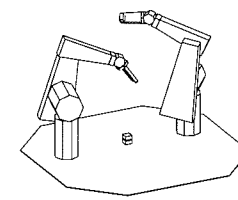
If all the grasping constraints cannot be met by the first pair in the list, the grasp planner tries, in turn, the other potential grasps.

A grasp face choice is shown in Figure 5.7. The position of the gripper in this figure has no significance except to indicate which grasp faces will be investigated further.

5.2.2 Projecting obstacles into the grasp plane

The selection of a grasp face pair establishes a grasp plane. This is shown in Figure 5.8. Recall that the grasp planner will plan motions that confine the gripper reference point to this plane.

Now consider the volume which is swept out as the gripper translates and rotates while confined to the grasp plane. In particular the highlighted rectangle in Figure 5.9(a) shows the volume (a **grasp volume**) that the rear gripper finger could sweep out. The only collisions the rear finger might suffer while moving about its grasp volume are with those portions of nearby objects that intersect this volume. Figure 5.9(b) shows the intersection of the rear finger grasp volume with local objects.



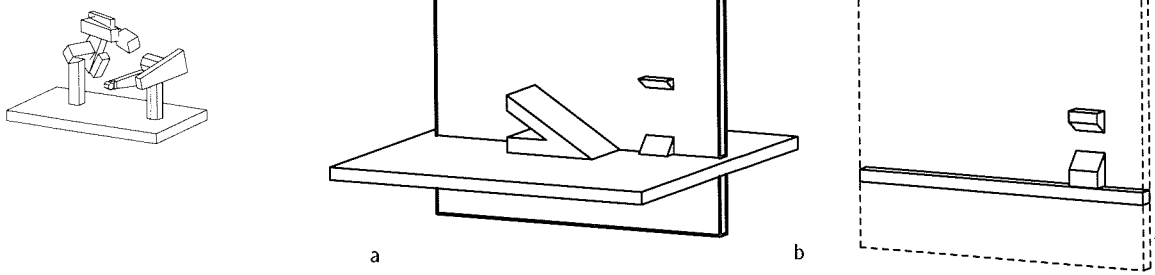
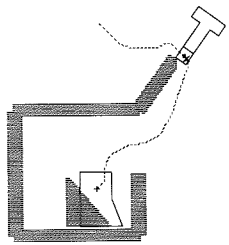


Figure 5.9
(a) Volume swept out by the rear gripper finger (highlighted), (b) portions of obstacles which collide with this volume.



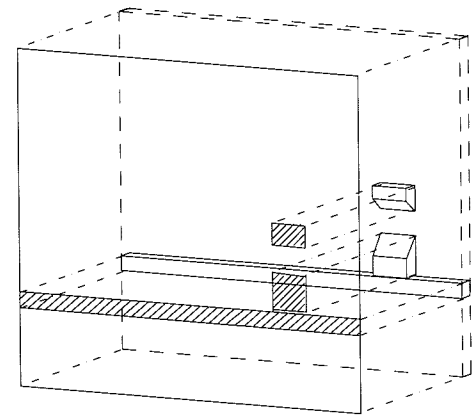


Figure 5.10
3-D obstacles are projected into 2-D obstacles (hatched) on the grasp plane.

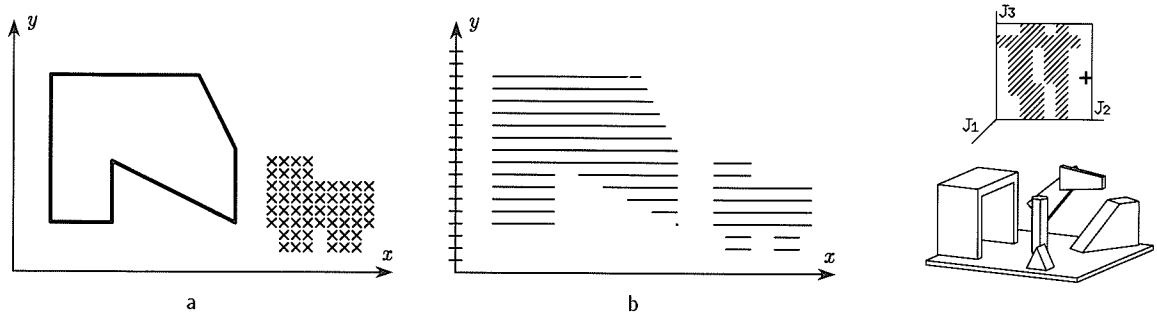
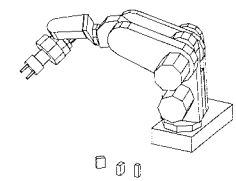
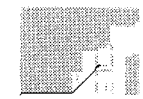


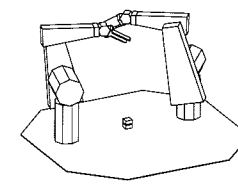
Figure 5.11
Scan lines represent two-dimensional objects. Conversion from (a) boundary or grid cell representation to (b) scan lines is straightforward.

At this point, the problem can be simplified with no further loss of generality by projecting the portions of objects which might cause a collision into the grasp plane (see Figure 5.10). This will greatly facilitate the next step—the computation of the C-space map for the gripper component. Note that because each gripper component can encounter different portions of obstacles as the gripper moves about the grasp plane, a separate grasp volume and separate projection must be made for each.



The grasp planner stores obstacle projections in the form of scan lines (see Figure 5.11). Scan lines are a common representation used in graphics to paint polygons onto a raster-scan device. There are several alternatives to scan lines for representing two-dimensional data; among these are boundary representations, quad-trees, and filled grid cells. The boundary and quad-tree representations may offer compact storage but the algorithms for dealing with them tend to be complex. The grid cell representation generally offers simple algorithms but may require more processing time than the others on a serial machine. Scan lines are a compromise between these extremes. Furthermore, conversion from boundary or grid cell formats (also used in HANDEY) to scan lines can be done rapidly.





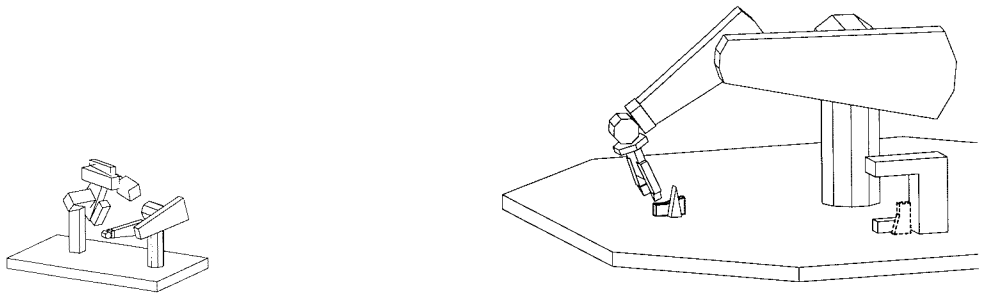
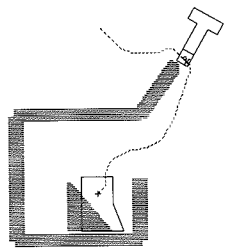


Figure 5.12
Pick-and-place example



Consider the pick-and-place example in Figure 5.12. Here the highlighted object must be moved to the position indicated by the dashed outline. Recall the gripper model used by HANDEY, (see Figure 5.2, page 111) and examine Figure 5.13. This figure shows the two-dimensional representation of the obstacles to the various gripper components at the pickup and putdown positions. (An outline of the pair of object faces chosen for grasping is superimposed for clarity.) In particular, note that the grasp object is hatched for the gripper flange and body projections but not for either gripper finger. This indicates that the grasp object is an obstacle for the flange and body but not for the fingers.

The scan line obstacle representation is also used to facilitate the ranking of grasp faces (see Section 5.2.1). Figure 5.14 helps to explain how this is done. The maximum area available for grasping is the intersection of the projection of the grasp faces onto the grasp plane. Some of this area is excluded because of blockages by nearby obstacles. Thus, the intersection of the grasp faces, and the free areas of both fingers at pickup and putdown provides a measure of the graspability of a particular face pair. When picking the grasp faces, this computation is made (at lower resolution) for each of the grasp face pairs of the grasp object.

5.2.3 Constructing C-space maps for the gripper

To determine the safe poses for the gripper moving in the grasp plane, an x, y, θ C-space map is constructed. Slices of this space represent different orientations of the gripper (compare Section 3.4.5). Since the grasp planner stores all two-dimensional data as scan lines it will be

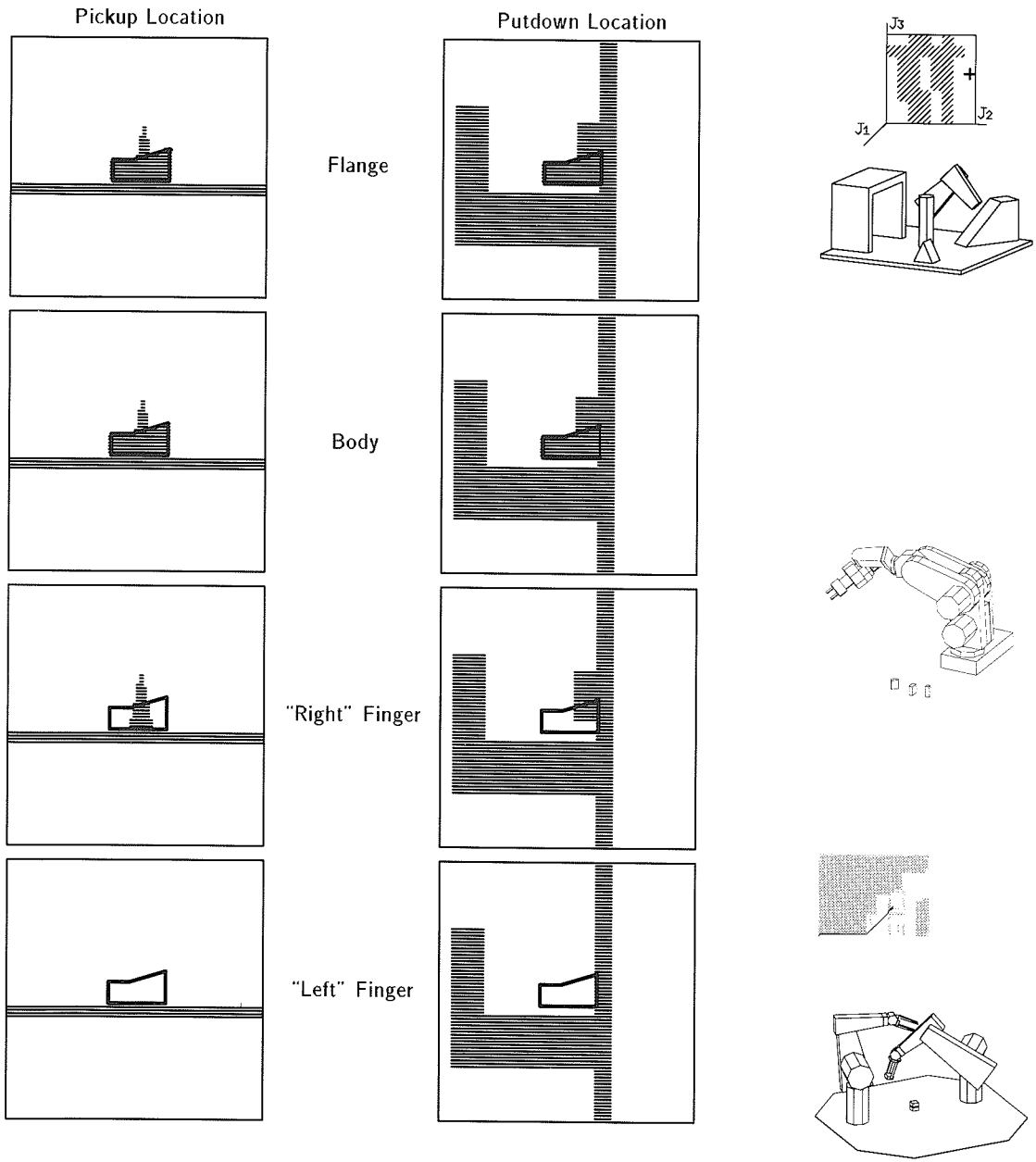


Figure 5.13
Obstacles to the gripper components

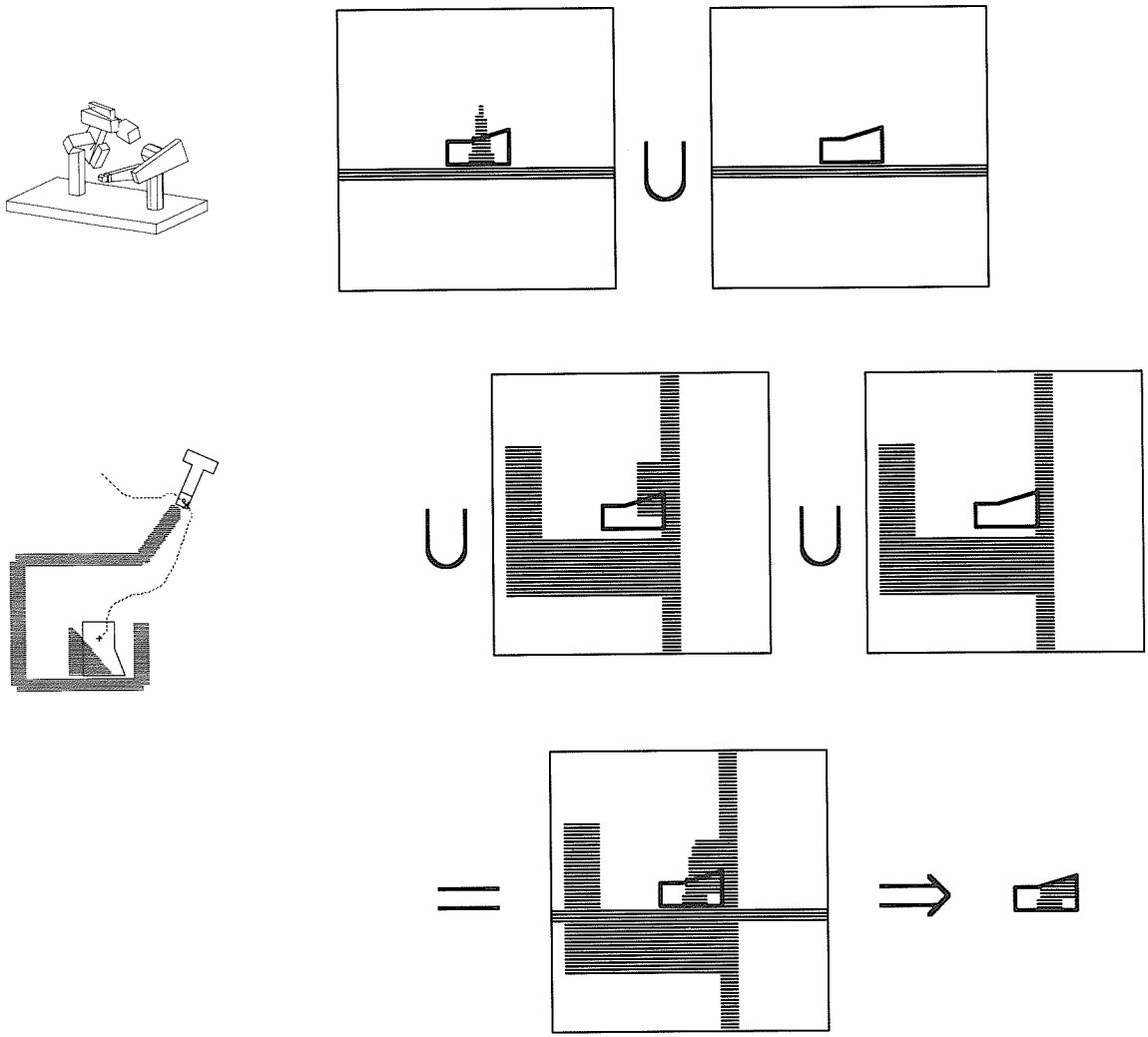
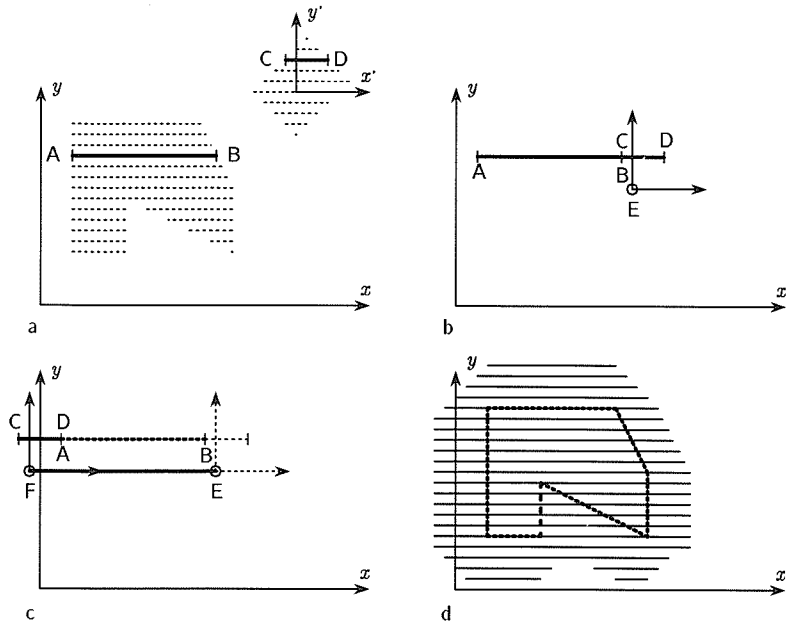
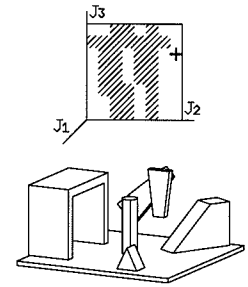


Figure 5.14
The graspable area is the non-hatched region on the grasp face pair. This is computed by taking the union of the obstacles to each gripper finger at pickup and putdown. The remaining non-blocked area on the face pair is available for grasping.





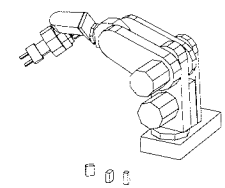
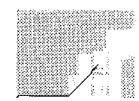
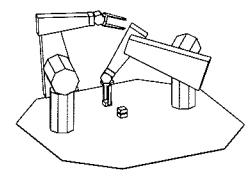


Figure 5.15
Computation of C-space scan lines. (a) One line, AB, of the stationary object combines with one line, CD, of the moving object. This produces, (b) and (c), a single scan line, EF, of the C-space obstacle, where $E = B - C$, and $F = A - D$. (d) All remaining scan lines.

necessary to compute the C-space map using scan lines. This operation is demonstrated in Figure 5.15.



The collection of scan lines to the left in Figure 5.15(a) represent a stationary object; those to the right (in the primed frame) represent the moving object (some component of the gripper). Computing the C-space obstacle yields the set of positions (also represented by scan lines) which are forbidden to the origin of the moving coordinate system. Consider the C-space obstacle generated by one line of the stationary object with one line of the moving object. Figure 5.15(b) and Figure 5.15(c) show the set of positions forbidden to the moving object origin. Figure 5.15(d) is the full C-space obstacle generated by computing the C-space obstacles of each stationary line with each moving line. The many overlapping lines thus generated are combined to simplify the representation.



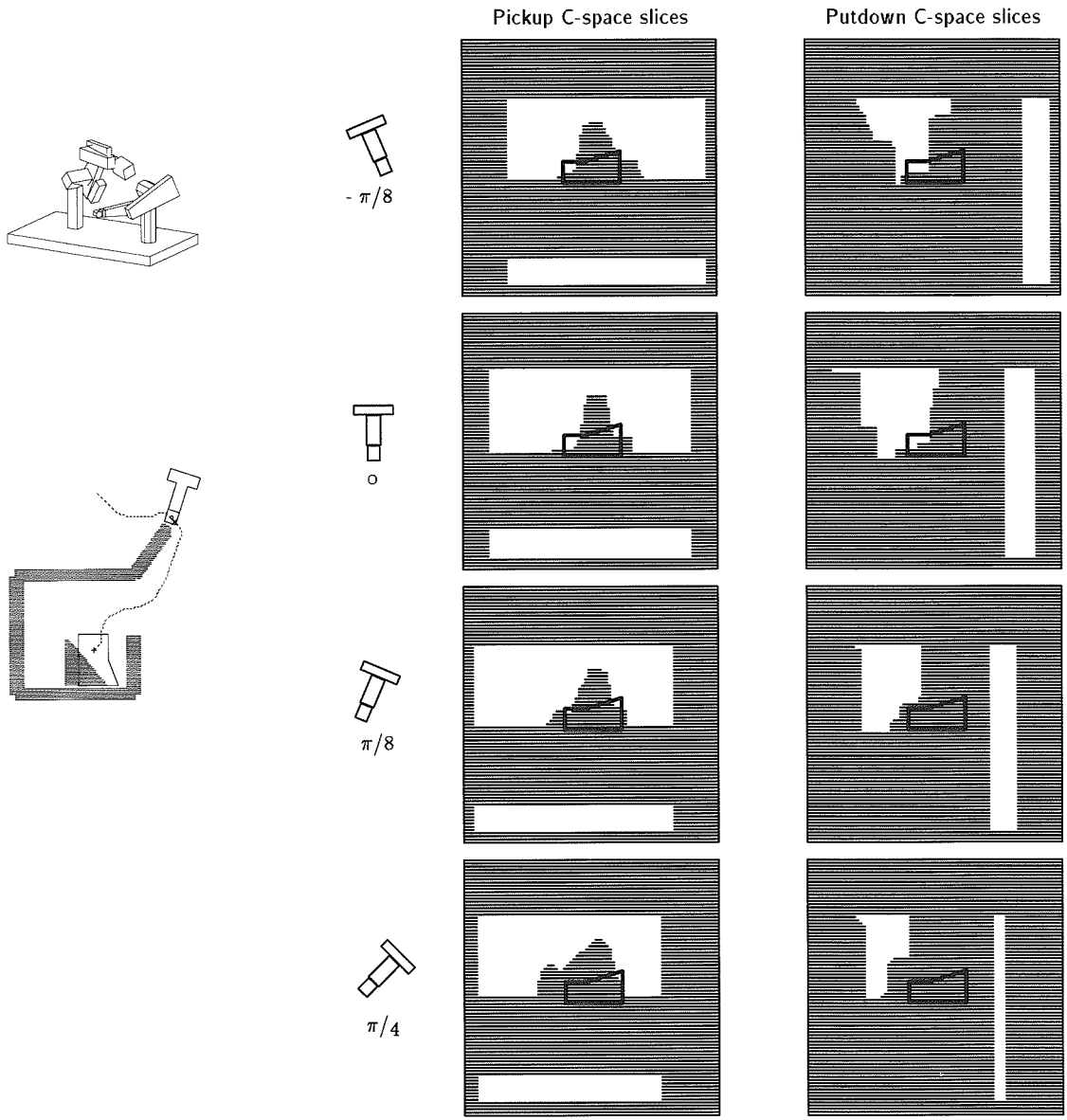


Figure 5.16
Pickup and putdown C-space slices for several orientations of the gripper

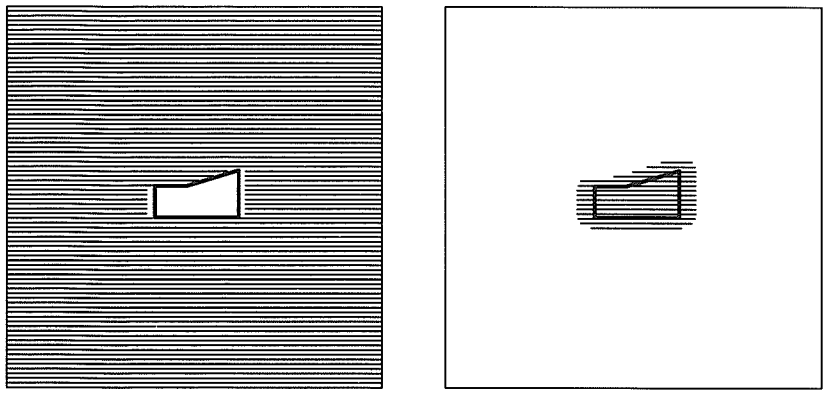
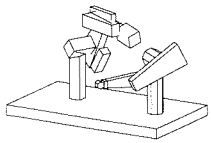


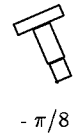
Figure 5.17
(a) Non-hatched area indicates all placements of the gripper reference point which produce viable grasps. (b) Non-hatched area indicates placements for which the gripper is clear of both grasp faces.

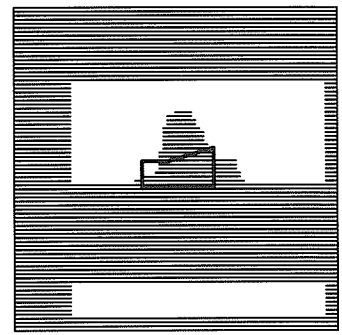
To generate the full C-space map for the gripper a calculation similar to the one outlined above must be carried out for each component of the gripper. This is illustrated in Figure 5.13, where the flange, body, and each finger of the gripper generate C-space obstacles at both the pickup and the putdown location. The obstacles in the C-space map for the gripper are the union of the obstacles in the C-space maps generated from each of the gripper components. An example of gripper maps for several gripper orientations[1] is shown in Figure 5.16.

It is now possible to simultaneously satisfy most of the constraints on the grasping operation. The non-hatched area of Figure 5.17(a) shows the gripper positions (regardless of orientation) which qualify as minimally acceptable grasps. The intersection of this area with the free area of one of the pickup C-space slices (for example Figure 5.18(a)) produces the set of gripper positions, for a particular orientation relative to the grasp object, that are acceptable grasps at the pickup pose. To find the set of gripper positions, for the selected gripper orientation, that also qualify at the putdown pose, we intersect the free area at putdown

[1] A new scan line representation is constructed for the gripper components for each orientation.







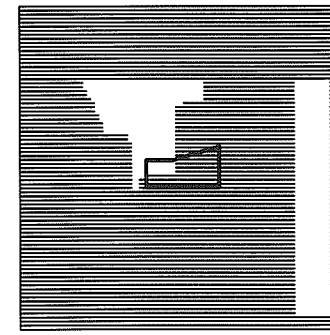
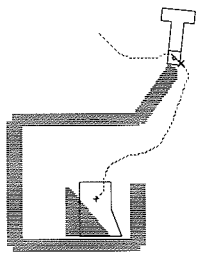
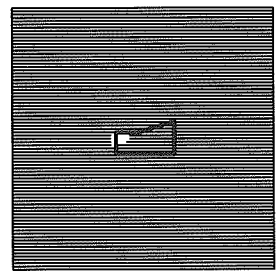
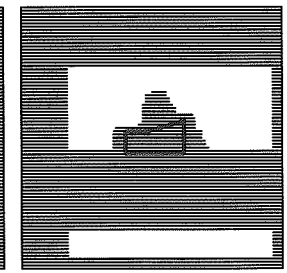


Figure 5.18
(a) Non-hatched area represents free C-space for given gripper orientation at the pickup location. (b) Non-hatched area represents free C-space for given gripper orientation at the putdown location







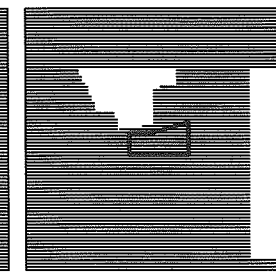
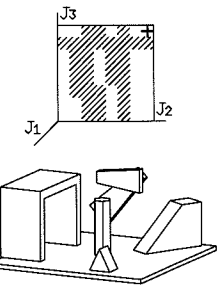


Figure 5.19
Considering only gripper collisions for orientation $\theta = -\pi/8$: (a) The set of points for which a grasp exists at both pickup and putdown. (b) The set of viable approach points. (c) The set of viable departure points.

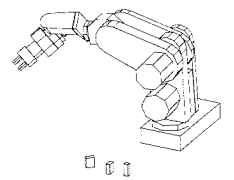
(Figure 5.18(b)) with the two areas already intersected. The result is shown in Figure 5.19(a). Also shown in Figure 5.19(b) and (c) is the set of viable approach and departure positions computed by intersecting the orientation-independent set of non-overlap points, Figure 5.17(b), with the free area at either pickup or putdown.



5.2.4 Arm constraints are C-space obstacles

The free regions in Figure 5.19 identify grasp, approach, and departure areas. Any point in the region avoids collisions between the gripper and nearby obstacles. This is not enough to guarantee a feasible grasp. As shown by the examples in Chapter 2 it is also necessary to avoid collisions involving the arm and to satisfy the robot's kinematic limits. We can deal with these arm constraints by identifying their effect on legal gripper poses in the chosen grasp plane.

The grasp planner checks each free point in the free areas of the grasp plane for both kinematic feasibility and for the presence of collisions. The inverse kinematics function for the robot is used to compute all the robot configurations that can place the robot gripper at a specific pose in the grasp plane. If no solution is possible at a particular pose, then that pose is marked as blocked. Kinematically feasible poses are then checked for arm collisions by checking the C-space map built by the gross motion planner. As we saw in Chapter 4, the gross motion planner builds a C-space map for the first three joints of the robot arm. Each pose in the grasp plane corresponds to some set of robot arm configurations (up to four for the Puma) in this three-dimensional C-space. Determining the presence of a collision simply requires a lookup in this map. The result of solving the inverse kinematics and looking up collisions in the arm's C-space map is a new set of obstacles that can be projected into the grasp plane. Doing this at the putdown location for the example produces the result in Figure 5.20(a). Figure 5.20(b) shows the effect due only to kinematic limits. Both of these figures are slightly misleading in that they show the result of the kinematic feasibility and arm collision test over the whole grasp plane. In fact, only the areas free of gripper collisions need be tested. This is usually only a small fraction of the grasp plane. Note also that a different such map will be obtained for each solution branch of the inverse kinematics. The solution branch with the most free space is the one of interest.



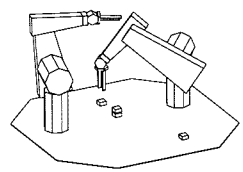


Figure 5.20
(a) Arm collisions at putdown mapped into the grasp plane (b) Arm kinematic constraints at the putdown pose mapped into the grasp plane

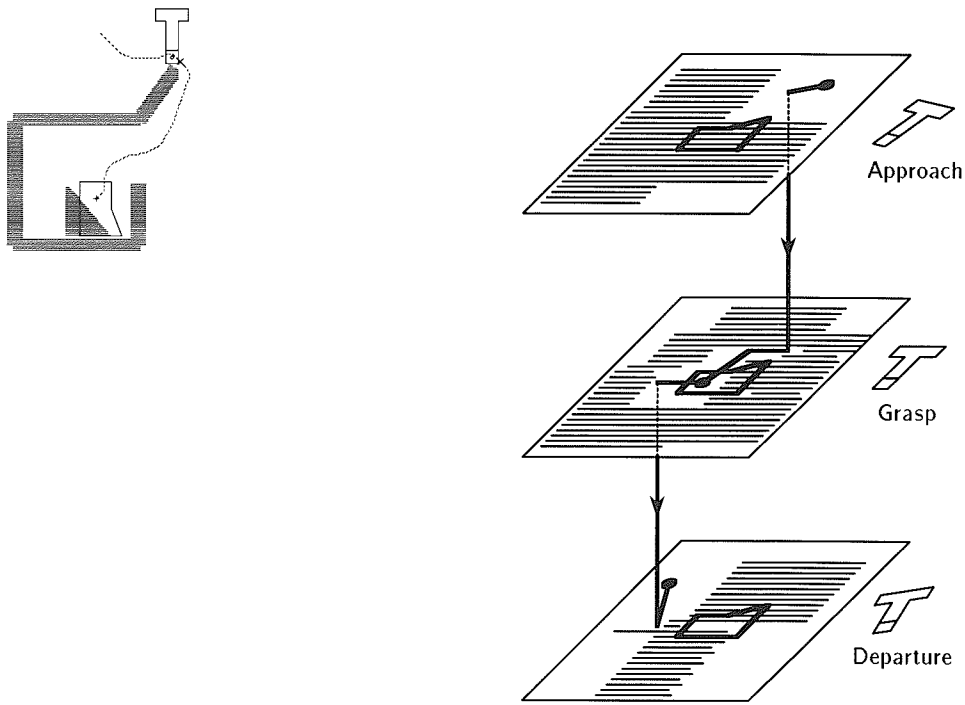
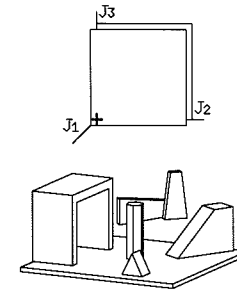


Figure 5.21
C-space search

5.2.5 Searching the C-space maps



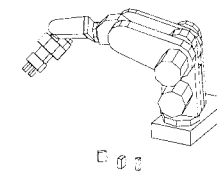
The discussion so far has focused on constructing C-space maps for a single gripper orientation relative to the grasp object. In general, we will need to construct slices of the C-space map corresponding to different gripper orientations. Our objective is to identify a path that starts in one of the legal approach regions, connects to a legal grasp region, and from there connects to a valid departure region. These regions need not be in the same gripper orientation slices, as illustrated in Figure 5.21.

To make precise how this search operates, we need a few definitions:

O The set of gripper positions that overlap the intersection of the grasp faces (see Figure 5.3, page 112).

N The set of gripper positions that do not overlap the union of the grasp faces (see Figure 5.4, page 114).

$F_A(\theta)$ The set of gripper positions, for gripper orientation θ, that are collision-free and kinematically feasible at the pickup (approach) pose (see Figure 5.18(a), page 126).

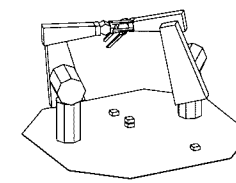


$F_D(\theta)$ The set of gripper positions, for gripper orientation θ, that are collision-free and kinematically feasible at the putdown (departure) pose (see Figure 5.18(b), page 126).

$G(\theta)$ The set of gripper positions in $F_A(\theta) \cap F_D(\theta)$ that are viable grasp points, where the fingers sufficiently overlap the grasp faces, Figure 5.19(a). $G(\theta) = F_A(\theta) \cap F_D(\theta) \cap O$

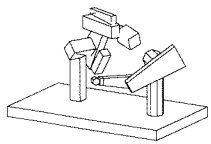
$A(\theta)$ The set of gripper positions in $F_A(\theta)$ that are viable approach points, where the fingers do not overlap the union of the grasp faces (see Figure 5.19(b)). $A(\theta) = F_A(\theta) \cap N$.

$D(\theta)$ The set of gripper positions in $F_D(\theta)$ that are viable departure points, where the fingers do not overlap the union of the grasp faces (see Figure 5.19(c)). $D(\theta) = F_D(\theta) \cap N$

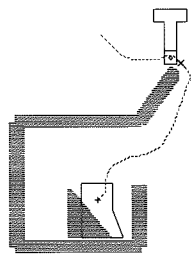


The objective of the grasp search is to find:

1. A grasp point $g \in G(\theta_i)$,
2. an approach point $a \in A(\theta_j)$,
3. a departure point $d \in D(\theta_k)$,
4. a path connecting a to g contained in $F_A(\theta)$ (θ may vary), and
5. a path connecting d to g contained in $F_D(\theta)$ (θ may vary).



The grasp planner proceeds by iterating over gripper orientation, θ, computing slices of both F_A and F_D. For each new θ slice, the planner updates the set of regions in the new slice that overlap regions in the previous slice. Each set of connected slices in F_A is tagged if it contains a viable approach point—if any of the slices intersects the O region. Similarly, the connected slices of F_D are tagged if they contain a viable departure point—if any of the slices intersects the N region. When each new slice is computed, the intersection $G(\theta) = F_A(\theta) \cap F_D(\theta) \cap O$ is checked. If $G(\theta)$ is not null, those connected regions of F_A and F_D overlapping G are marked as containing a grasp point and a link is established between the regions. The search stops when a region of F_A contains both an approach and a grasp point and is linked to a region of F_D that contains a departure point and the same grasp point.



Given these connected regions, a path is found connecting the "central point" of the approach, grasp and departure regions. These central points are computed by a technique, borrowed from binary vision [31], called **thinning**. This technique consists of successively stripping off the perimeter of an area until only a single central point remains. Another variation on thinning is used to find paths connecting the designated approach, departure and grasp points to the central points in the intersection between regions in adjacent slices.

5.3 Using depth data in grasp planning

One of the earliest applications of HANDEY was to pick up objects localized by processing a **depth map** [48]. A depth map is a two-dimensional array whose indices code for position and whose elements represent the height of a surface at that location. HANDEY uses a triangulation-based laser range-finder to construct depth maps. Figure 5.22 shows a set of objects and Figure 5.23 shows the corresponding (simulated) depth map.

The use of a depth map as a source of information about the environment presents some new problems for the grasp planner. We have assumed up to now that every object in the robot's workspace is represented by a polyhedral model. To preserve that assumption in the context of the use of a sensory system, one can either require that the user specify the pose of every nearby object save the one to be localized, or the recognition system must recover the identity and pose of every

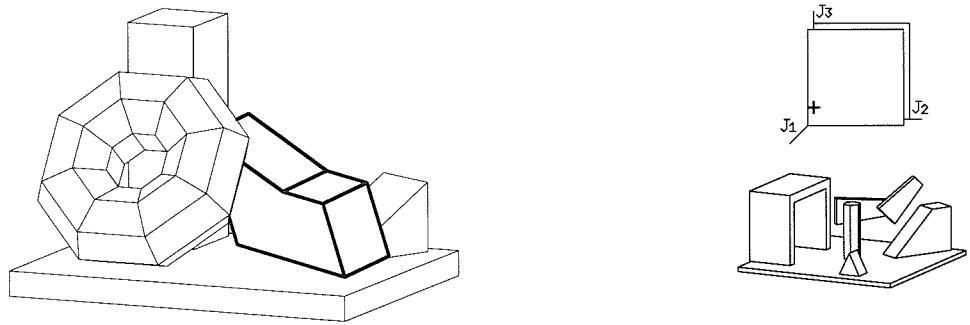


Figure 5.22
A grasp object (highlighted) among obstacles

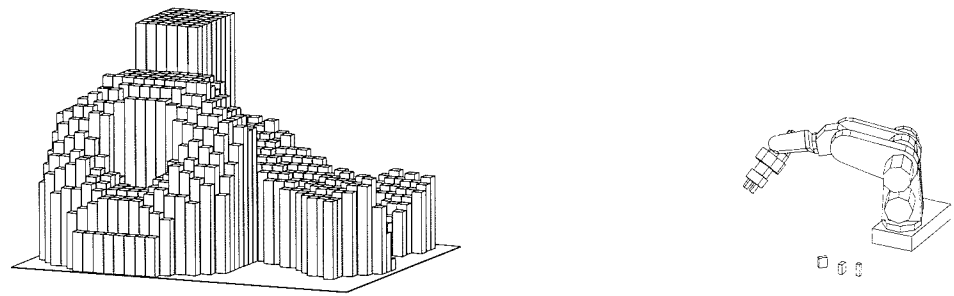
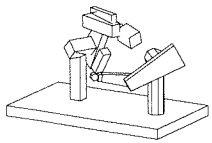


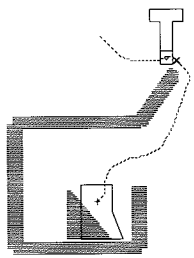
Figure 5.23
Simulated depth map of the preceding scene

object in its field of view. We chose, instead, to relax the polyhedral model assumption within the context of the grasping operation.

To plan a grasping operation, the system need only identify the object to be grasped from the depth map. The height values not associated with the grasp object can be treated as uninterpreted clutter. For the grasp planner to use this data, it must detect potential collisions between the measured surfaces and the various gripper components: This is done by testing each point (pixel) in the depth map within the footprint[2] of the grasp volume of a gripper component. If any portion of the ray

[2]The footprint of a grasp volume is its projection onto the horizontal plane of the depth map.





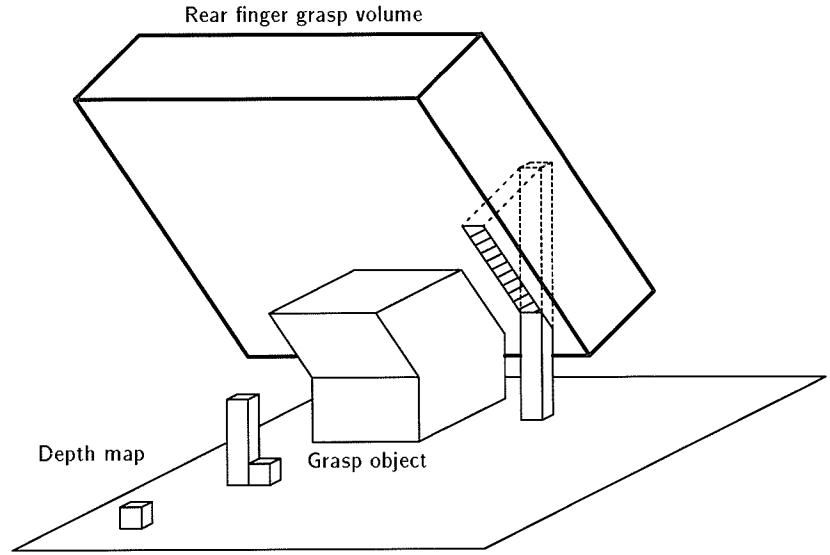


Figure 5.24
A depth map pixel (which contains altitude information) intersects the grasp volume of the rear finger. The intersection is projected onto the grasp plane.

extending downward[3] from that point intersects the grasp volume then that line segment is projected onto the grasp plane. This procedure is illustrated in Figure 5.24.

There are, however, some difficulties. Depth map data points produced by the grasp object are indistinguishable from those produced by unrecognized clutter objects (see Figure 5.25). The darkened area represents a point in the depth map contributed by the grasp object. Since a ray extending downward from this point intersects the front finger's grasp volume, the previously described projection scheme would require projecting this point onto the grasp plane. In that case an overhanging grasp object face would always be seen as completely blocked. This cannot be allowed. Therefore, after the grasp object has been identified in the depth map and prior to the intersection of depth map points with the grasp volumes, the points for which the grasp object is responsible are removed from the depth map—their height values are set to zero.

[3]Downward means toward the horizontal plane of the depth map.

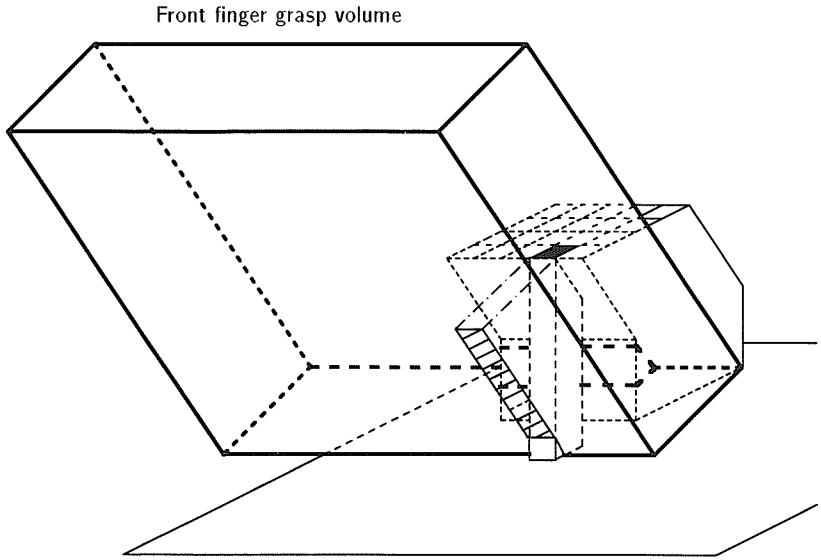


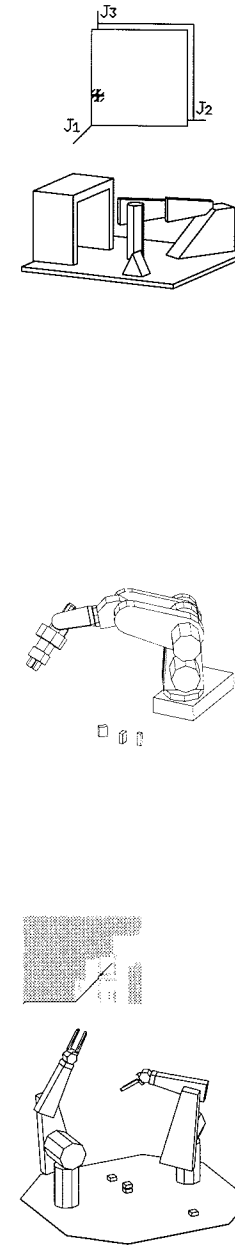
Figure 5.25
Some depth map pixels correspond to overhanging portions of the grasp object. Projection of such points onto the grasp plane must be forbidden.

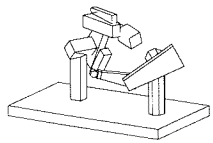
Note that the possibility remains that a non-modeled clutter object could be hidden under an overhanging part of the grasp object. This condition cannot be determined by a single depth map from a fixed point of view; it can be detected only at runtime, possibly by a guarded move. HANDEY currently ignores this possibility.

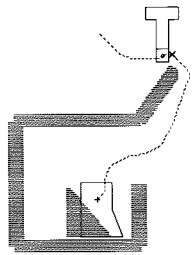
In addition to intersecting depth map data with the grasp volumes, it is still necessary to intersect nearby modeled objects with the grasp volumes and project that information as well. In this operation, we must also include parts of the grasp object that protrude beyond the grasp faces since these can also cause collisions.

5.4 The potential-field planner

The grasp planner described above, based on configuration space maps, evolved from an earlier planner which used a potential-field motion planning method [35, 36]. In this section we will briefly review the design







of this earlier grasp planner both because it illustrates a different set of motion planning techniques and because it sheds some light on the interactions among the constraints in pick-and-place planning.

Many of the basic assumptions of the grasp planner described above were inherited from the earlier grasp planner. In particular, both systems use a two-dimensional grasp plane on which the three-dimensional obstacles are projected. Also, both systems divide the problem into the search for an approach and a departure path. The two planners differ in the method of choosing the approach, grasp and departure points as well as the method of finding the paths between them. In the C-space-based planner, the choice of approach, grasp, and departure points and the choice of paths is intertwined, driven by the connectivity of the C-space. In the earlier potential-based planner, the system guesses approach, departure and grasp points and then searches for paths, using a potential field, to connect them. Specifically, the earlier planner planned the approach and departure phases nearly independently.

The following steps were used to plan the approach phase:

1. Find an approach pose on the grasp plane near the pickup location using a generate-and-test strategy: test several candidate approach positions and orientations for gripper collisions, arm collisions, and kinematic feasibility.
2. Identify a region on the grasp faces which is not totally blocked by obstacles at pickup and putdown, then shrink this region to a single point, the nominal grasp point.
3. Plan a path from approach to grasp using a potential-field method.

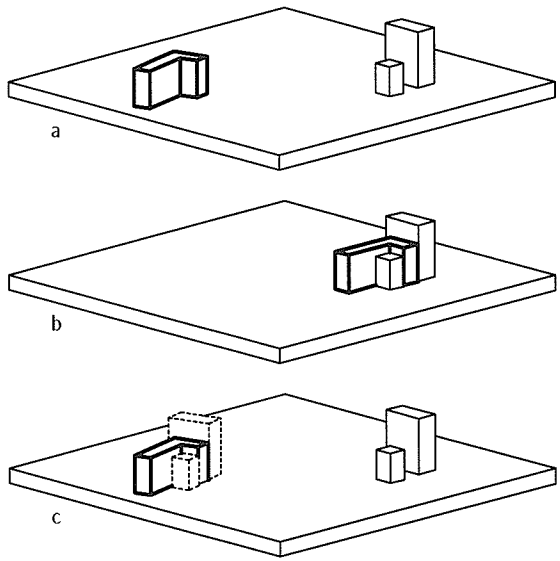
These steps were used to plan the departure phase:

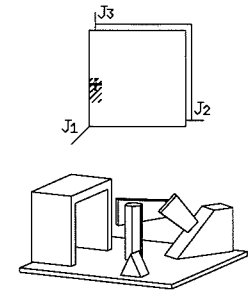
1. Find a departure pose on the grasp plane near the putdown location using a generate-and-test strategy.
2. Plan a path from grasp to departure using a potential-field method.

The key difficulty with this earlier planner, as we will see, proved to be in the initial guessing of approach, departure and grasp points.

5.4.1 Obstacle backprojection

The earlier planner enforced the constraints requiring a grasp to be collision free and kinematically feasible at both pickup and putdown in an





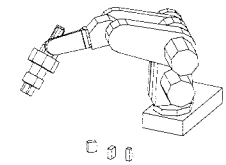
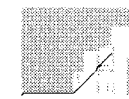
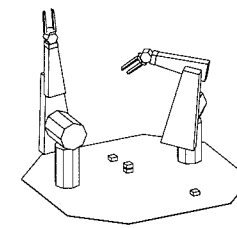


Figure 5.26
Using backprojection to satisfy pickup/putdown collision constraints: (a) shows the initial world configuration; the highlighted object must be positioned as in (b). In (c), the putdown obstacles have been backprojected into the pickup location.

overly conservative way. The key idea is called **obstacle backprojection**. The portions of the grasp faces that will be unblocked at both pickup and putdown can be identified if obstacles near the putdown position are treated as though they were present at the pickup location, appropriately rotated so as to maintain their relationship to the grasp object (see Figure 5.26).

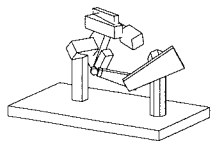


If, while planning the approach path, the grasp planner ensures that at all times the gripper avoids both the objects at the pickup pose and the backprojected obstacles from the putdown area, then the final position and orientation will leave the gripper collision-free at both pickup and putdown. However, it is not necessary that the gripper avoid backprojected obstacles while *approaching* the grasp point, only that its motion not *terminate* while in collision with one. The use of a potential-field based planner limits our ability to incorporate such a refined constraint, since one can't predict when the potential-field planner will terminate

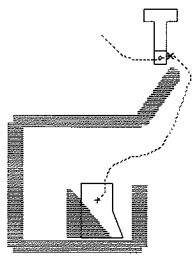


the motion. This was an important way in which the earlier grasp planner overconstrained the grasping problem.

5.4.2 The potential-field method



The initial motivation for basing a grasp planner on a potential-field method was to exploit its ability to find simple paths very quickly. We understood that these methods are susceptible to stopping short of the goal because of local minima in the potential field. Our initial approach, described in this section, ignored the local minima problem. Our experience with the limitations of this method then led us to refine our approach to attempt to minimize the impact of local minima (see Section 5.4.3).

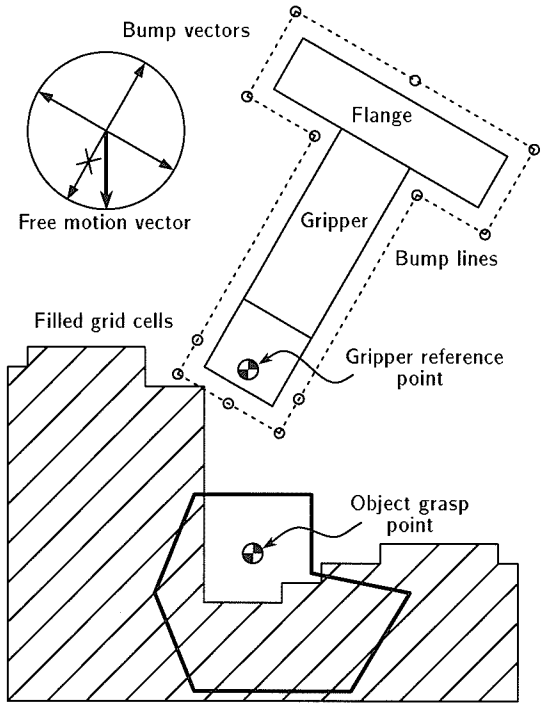


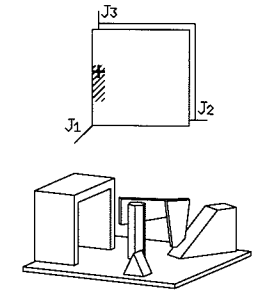
We require a path from a nearby approach point to the grasp. Figure 5.27 shows one iteration in the computation of this path. The gripper attempts to move its reference point toward the grasp point while avoiding obstacles. To avoid the obstacles, the gripper is surrounded by bump lines at some distance, d. A **bump line** is a line segment which moves with the gripper on the grasp plane and is checked each iteration for collisions with filled grid cells. A **bump vector** is a unit vector perpendicular to a bump line pointing away from the gripper. Because the edges of the gripper model are aligned with the coordinate axes of the gripper and the bump lines are parallel to the components of the model there are only four distinct bump vectors.

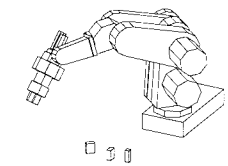
In the absence of intervening filled grid cells, the motion of the gripper is a simple translation along the vector connecting the gripper reference point with the object grasp point. The unit vector in this direction is called the **free motion vector**.

During any iteration when a bump line/grid cell collision is detected, the motion of the gripper must be restricted. In this case motion is allowed only along one of the bump vectors. After bump vectors associated with colliding bump lines have been eliminated, a motion step is executed along the bump vector most closely aligned with the free motion vector. In Figure 5.27, the next motion step will occur along the bump vector pointing toward the lower right.

This method is analogous to a potential field with a very strong dependence on distance. Beyond the distance d, the repulsive force is zero; within d the force is sufficiently strong to prevent motion in a direction that would decrease the distance.







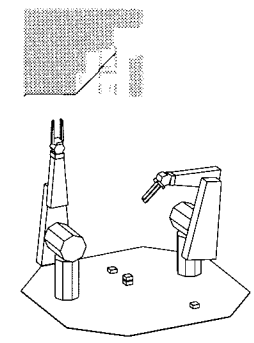
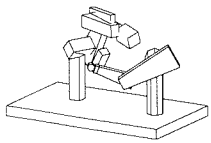
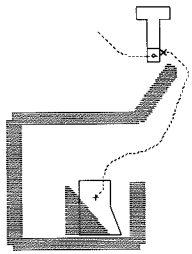


Figure 5.27
A potential-field method is used to plan the path to a grasp. During a collision between a bump line and an object, motion in the direction of the corresponding bump vector is forbidden (indicated by an '×'). Bump lines are constructed such that the sign of the torque does not change along their length. Small circles indicate the end points.

It may be necessary to change the orientation of the gripper as it moves toward the grasp point. Bump lines provide a way to do this as well. Any colliding bump line produces a torque about the gripper reference point whose magnitude is proportional to the cross product of the bump vector and a vector connecting the gripper reference point with the center of the bump line. Bump lines must be constructed such that the sign of the torque doesn't change along the line. The total torque on the gripper is then the sum of the torques generated by each colliding bump line. In Figure 5.27, the next rotation increment will be clockwise.





```
procedure potential-field
begin
  for i from 1 to max-iterations
    begin
      if bump-lines do not collide with filled grid cells
        then position = position + k × free-motion-vector
      else
        begin
          Compute nearest-bump-vector
          Compute total-torque from the bump lines
          position = position + k × nearest-bump-vector
          orientation = orientation + m × total-torque
        end
      end
  end
```

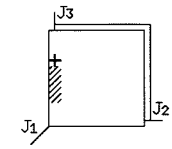
Figure 5.28
Original potential-field algorithm. The *free-motion-vector* is the unit vector from the gripper reference point to the grasp point. The *nearest-bump-vector* is the bump vector most closely aligned with the *free-motion-vector* after the bump vectors associated with colliding bump lines have been eliminated. Constants k and m are scaled such that no point on any bump line can move a distance greater than the width of one grid cell during one iteration.

Each iteration, the gripper translates and rotates in a way determined by the direction to the grasp point relative to the gripper and by near collisions with obstacles. The rotational and translational step sizes are chosen such that no point on any bump line is moved by more than one grid cell per iteration. This precludes the possibility that any filled grid cells will penetrate the bump lines.

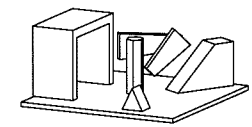
The motion terminates successfully when the gripper reference point is within some predetermined distance of the grasp point or when the gripper fingers overlap the grasp faces by a sufficient amount. An unsuccessful termination occurs when no path has been found after some set number of iterations. Lack of progress toward the goal can be another condition of failure. This whole procedure is described by the pseudo-code in Figure 5.28.

The method outlined above does not satisfy all the required constraints on a viable grasp. At some point in the robot path implied

by the computed gripper path, one or more of the following conditions may occur:

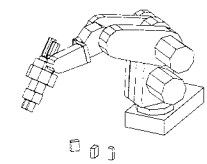


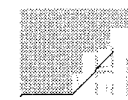
1. the arm is in collision with an object in the environment,
2. there is no kinematic solution, or
3. there is no continuous kinematic solution—moving from the previous to the current path step requires a reconfiguration of the robot.

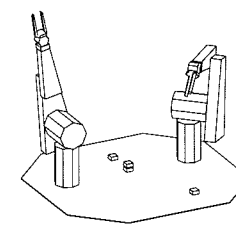


In our first implementation, paths found by the algorithm in Figure 5.28 were subjected to verification to ensure that the arm collision and kinematic feasibility conditions were met. In practice, failures at this point were frequent. In subsequent implementations, robot path verification was incorporated into the inner loop of the potential field method. If the motion step computed from local considerations failed in one of the ways listed above, then a step along a different bump vector or a small arbitrary rotation was tried.

After successfully planning a grasp, the system still experienced frequent failures at the putdown step because the grasp was chosen without regard to the kinematic limits at the putdown position. To address this problem, we used an approach similar to obstacle backprojection. Just as the grasp planner avoided the backprojected obstacles from the goal at every step of the approach (so as to guarantee that the approach path did not terminate in a goal collision), it would now perform the kinematic/arm-collision test for both the pickup and putdown pose at every step on the approach path. This approach highly overconstrained the problem since the kinematic/collision constraint really applied only to the final point of the path. Nevertheless, this approach still managed to find solutions for many of the pick-and-place problems tried. Partly, this is because requiring the existence of a series of collision-free and kinematically feasible solutions near (and at) the putdown pose increased the likelihood of later finding a gross-motion to reach the putdown pose and a departure path from there.

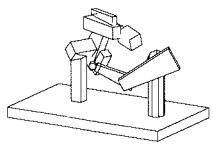






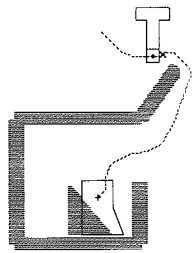
5.4.3 Improved potential-field strategy

It is well known that motion planners based on simple potential methods can fail by becoming trapped in local minima (see Figure 5.29(a)). The attractive force driving the motion always points toward the goal; no motion away from the goal is possible. The moving object will be



trapped when, due to the repulsive forces from obstacles, the net force has no component toward the goal.

We found that a simple modification can avoid one kind of local minima. If the start and goal points are known and lie in a common region of the free space, it is possible to construct a path between them. One method for doing this is as follows. While preserving the start and goal points and the topology of the free space (represented by empty grid cells), shrink the free space by removing cells from its boundary until a step is reached when no cells can legally be removed. This is the thinning operation mentioned in Section 5.2.5. At this point there will either be one or more filaments connecting start and goal (see Figure 5.29(b)) or there will be no connection. Lack of a connection proves that no path exists; it will be necessary to try different grasp faces. If a filament is found, however, it can be used to drive the motion of the gripper.

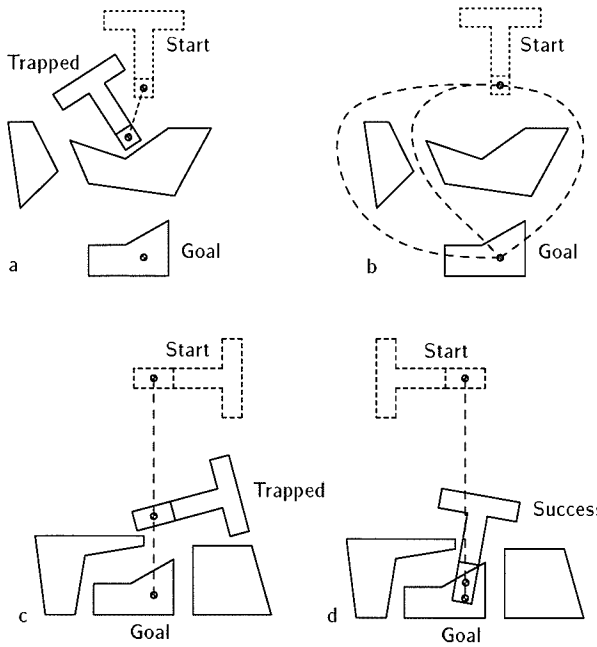


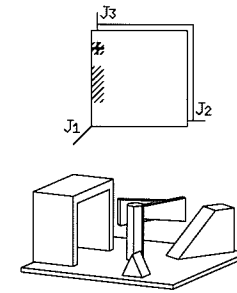
The modifed potential-field path planner proceeds as follows:

1. Grow all obstacles by half the width of the finger. (This discourages motion through too narrow channels.)
2. Construct a filament path by thinning the free region containing the start and the goal. (Select the shortest path first if more than one filament is discovered.)
3. Place the attraction point at the starting point of this path.
4. In each iteration, move the attraction point along the filament toward the goal.
5. Use the existing potential-field mechanism to generate torques and small displacements for the gripper to move it out of the way of obstacles as it progresses toward the goal.

An important refinement of this approach is that the attraction point must not move forward along the filament after any iteration when the gripper did not move toward the attraction point. Otherwise, the attraction point and gripper reference point become separated and the algorithm degenerates to the one previously described. Left-Margin Movie 2 shows a path found by this planner. (See the description of the Margin Movies in "On the Margins," page xvi.)

This method is superior to the simpler one in two important ways: it can sometimes prove that no path exists, and it solves a broader class of problems. Unfortunately, it can still fail to find a viable path even when one exists. This is because, although the path construction





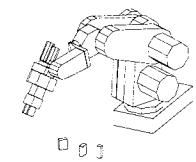


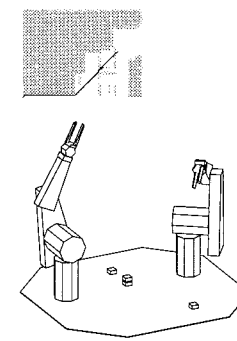
Figure 5.29
Partial solution to the local minima problem. (a) Local minimum traps gripper from some starting locations. (b) Precomputing path avoids local minima in Cartesian space. (c), (d) Local minimum in configuration space can trap gripper from some starting configurations.

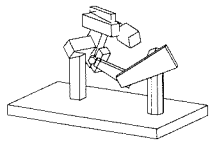
step eliminates local minima from the Cartesian space, minima may still remain in the gripper's configuration space. See Figure 5.29(c) and (d).

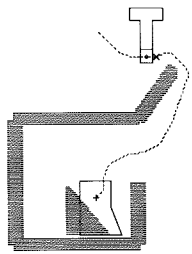
The many refinements to the potential-field grasp planner have improved its performance substantially. However, two fundamental shortcomings of the potential-field method remain:

1. it cannot guarantee to either plan a grasping operation if one is possible or halt if one is not, and
2. it cannot incorporate certain important constraints in other than a significantly overly conservative way.

For these reasons we developed the C-space grasp planner. It has performed satisfactorily.

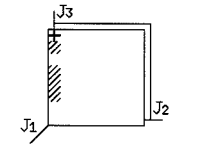


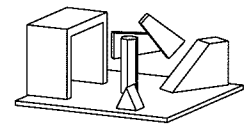




6 Regrasp Planning

Regrasping must be done whenever a robot's grasp of an object is not compatible with the task the robot must perform. Consider, for example, a robotic cell with a manipulator alternatively picking up parts from a conveyor or a pallet and inserting them into an assembly. Assume that the parts are presented in arbitrary orientations. The task may not be achievable with a single grasp because none of the grasps feasible at the pickup may be feasible at the assembly (see Figure 6.1). There is no single grasp to pick up the part A and place it in the assembly near the part C; the part must be grasped, then it must be placed away from part B and it must be *regrasped* before performing the final assembly.

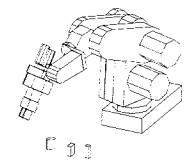


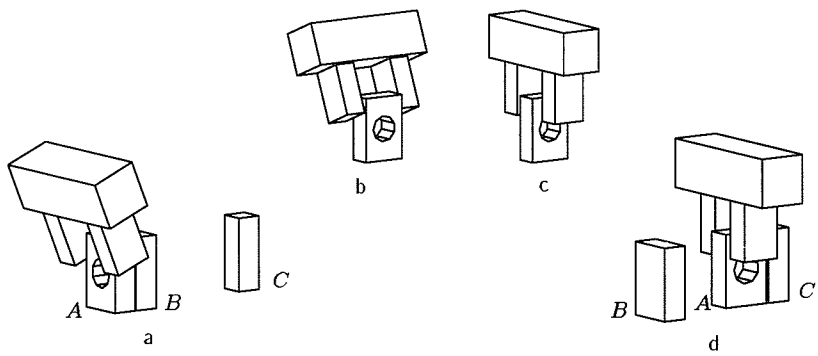


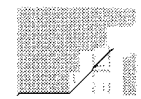
The regrasp planner in HANDEY is invoked as follows:

Regrasp($goal, robot, world$)

When this planner is called, the robot must be holding the part which is specified in the goal. The planner will generate an appropriate sequence of pairs of robot operations consisting of placing the part on the table (or in another gripper) and regrasping it with another grasp until a grasp compatible with the putdown pose in the goal is found. In this chapter, we will first focus on the use of a table for intermediate placements. We will explore the use of a second gripper in Section 6.7.







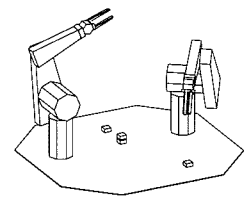
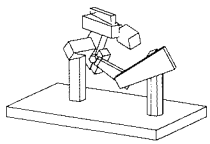
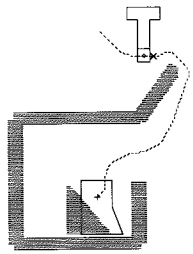


Figure 6.1
In order to move part A from its initial position in (a) to its destination in (d) it is necessary to move it to an intermediate position (b) and regrasp (c).



In our approach to regrasping using a support table, we characterize the possible grasps of an object and its stable placements on the table. Both grasps and placements are described by the discrete choice of a surface of contact, combined with the choice of a continuous rotation parameter. The regrasping problem is then solved by computing transitions in a space where we represent all compatible conjunctions of grasps and placements. A grasp and a placement are **compatible** if neither the grasp nor the placement causes a collision and the placement is kinematically feasible given this grasp of the object.

If there were no constraints due to the presence of nearby objects or due to joint angle limits, all regrasping could be done in a single step. In practice, these constraints may require that more than one regrasping step be done. The Right-Margin Movie 2 shows a sequence of grasps and placements produced by the HANDEY regrasp planner (See the description of the Margin Movies in "On the Margins," page xvi.)



6.1 Grasps

The regrasp planner makes the same set of basic assumptions on legal grasps as the grasp planner does (Section 5.1). In particular, grasps will involve contact between the interior surfaces of the fingers, the **grip surfaces**, and nearly parallel grasp faces, which define a grasp plane.

We will build a subset of this type of grasp by considering the union of several grasp classes: $\{\mathcal{G}_i\}_{i\in\{1,\ldots,n\}}$. Each **grasp class** $\mathcal{G}_i$ is characterized by a **grasp plane** and a **grasp point**. The grasp plane is always parallel to and midway between the grasp faces of the object. The grasp point lies always in the grasp plane and can be projected inside the two faces defining the grasp plane.

By restricting a particular point attached to the gripper, named the **gripper reference point**, to coincide with the grasp point, and by restricting the motion of the gripper to be such that the grip surfaces stay parallel to the grasp plane, we can define a family of grasps corresponding to this grasp class.

We associate each grasp class $\mathcal{G}_i$ with a coordinate frame attached to the object and written $\mathbf{G}_i$. The grasp frame $\mathbf{G}_i$ is defined as follows: its origin G_i is the grasp point, the vector z_{G_i} is normal to the grasp plane,

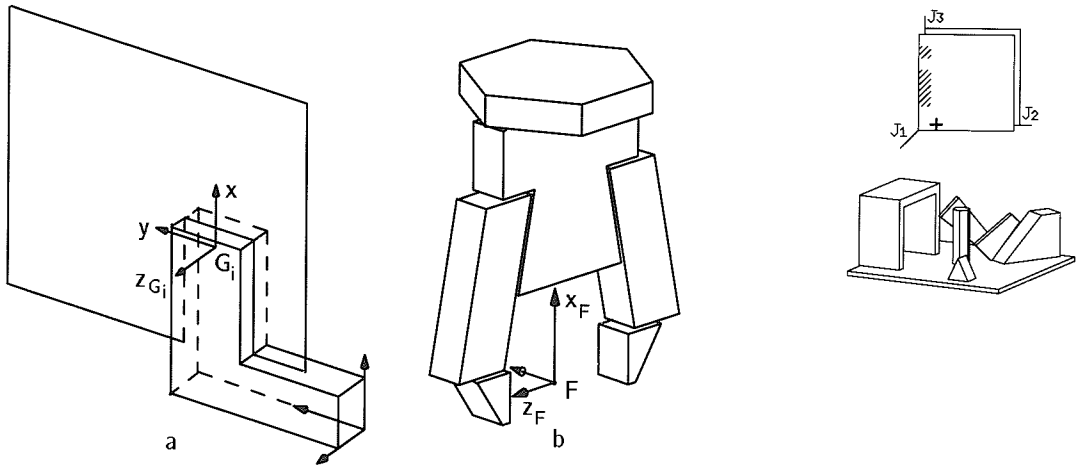


Figure 6.2
(a) Definition of the grasp frame. (b) Definition of the finger frame.

the x- and y-axes lie in the grasp plane. The orientation of the x- and y-axes is chosen arbitrarily (see Figure 6.2(a)).

We also define a coordinate frame attached to the gripper, called the **finger frame**.[1] The finger frame $\mathbf{F}$ is defined as follows: vector z_F is normal to the grip surfaces. Vector x_F points towards the wrist of the robot. The origin of the finger frame, F, is located near the end of the fingers and is midway between the grip surfaces. It is the gripper reference point (see Figure 6.2(b)).

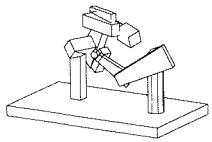
By restricting the type of grasps, the relative position of the grasp frame and the finger frame can be parameterized by a single parameter, θ, corresponding to the angle between x_F and x_{G_i} (see Figure 6.3).

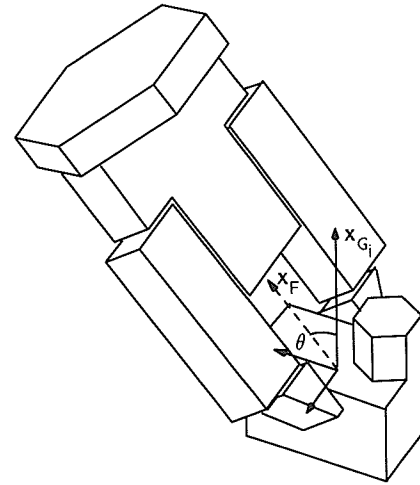
Since the finger frame is attached to the gripper, the relative position of the gripper and the object is totally determined by G_i and θ:

$$\mathbf{G}_i\mathbf{R}_z(\theta)\mathbf{F}^{-1} \tag{6.1.1}$$

where $\mathbf{F}$ denotes the relative position of the finger frame to the wrist

[1] The regrasp planner and the grasp planner choose slightly different definitions of the finger frame. Since they are independent modules each can choose a definition for the finger frame which suits its own purposes.





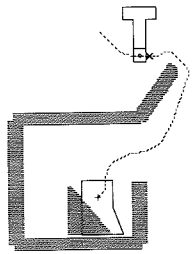


Figure 6.3
Parameterizing a grasp by $\mathbf{G}_i$ and θ

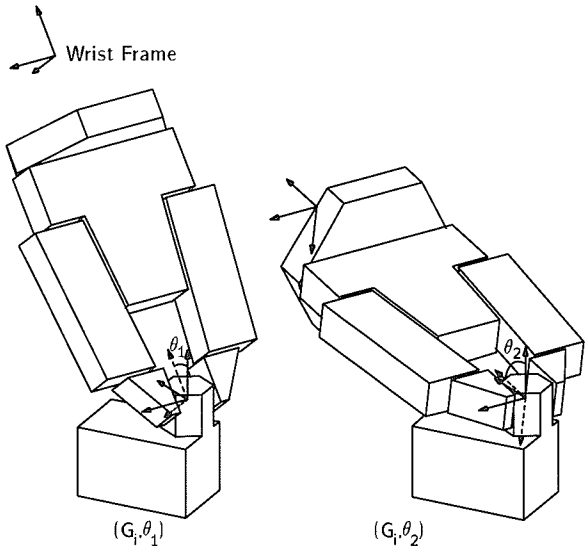
frame and $\mathbf{R}_z(\theta)$ a rotation about the z-axis. We denote a particular grasp from the grasp class $\mathcal{G}_i$ as:

(G_i, θ)

Figure 6.4 shows two grasps which belong to the same grasp class.

The set of grasp classes is obtained as follows: For each (convex) edge of the object we define two potential contact points. These points are located on the two adjacent faces defining the edge. They are obtained by displacing the middle point of the edge towards the interior of the faces. For example the edge e defines two potential contact points, c_1 and c_2 (see Figure 6.5(a)). A potential contact point generates a grasp class if it can be projected inside another face of the object parallel to the face to which it belongs. In this case, the grasp plane is defined as the plane parallel to and midway between these two faces. The grasp point is obtained by projecting the potential contact point onto the grasp plane.

Note that it may appear as if this method generates many redundant grasps, for example, in a cube, each grasp point is generated twice, one for each of the corresponding edges of opposite faces. In fact, these points



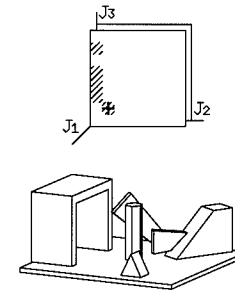
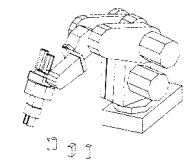
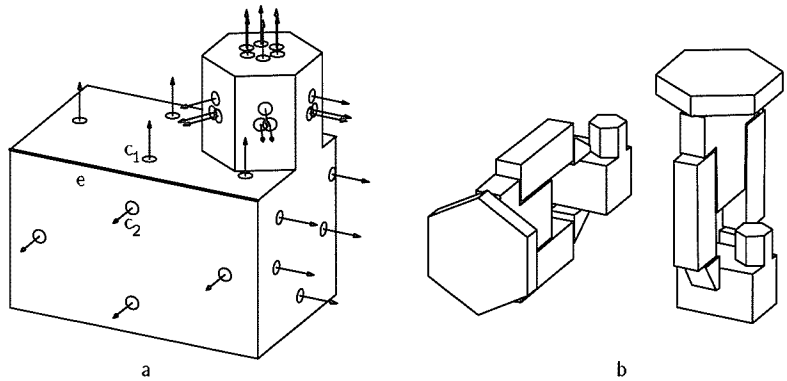


Figure 6.4
Two grasps of the same grasp class





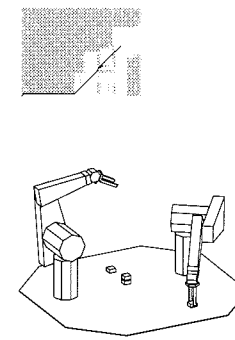


Figure 6.5
(a) Definition of the contact points. (b) Two grasp classes associated with one edge

are associated with different normals and correspond to grasps in which the gripper is rotated 180° about its x axis. It is the case, however, that small faces will generate many grasp points clustered together, such as those on the upper faces in Figure 6.5(a). These points can be readily eliminated.

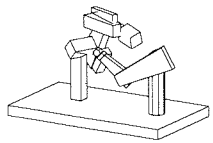
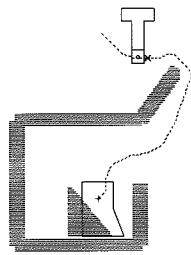


Figure 6.5(b) shows the two grasp classes for one edge. For each corresponding grasp class, the gripper is shown in the $\theta = 0$ orientation.

6.2 Placements on a table

A **placement** of a three-dimensional polyhedron can be characterized by a face of its convex hull[2] in contact with the table. Such a placement is stable if the projection of the object's center of mass onto the table lies in the interior of the support face. We denote by $\mathcal{P}_j$ the class of stable placements sharing the same contact face with the table. In this section, we will build a subset of all the possible placements by considering the set of several of these placement classes: $\{\mathcal{P}_j\}_{j\in\{1,\ldots,m\}}$.



Each **placement class** $\mathcal{P}_j$ is characterized by a **placement plane** and a **placement point**. The placement plane is supported by the face of the convex hull which is in contact with the table. The placement point is defined by the projection of the object's centroid on the placement plane. (In HANDEY, we do not model the density of the various components of the object, therefore, we cannot accurately compute the center of mass of an object. Instead, we make the implicit assumption of uniform density and use the centroid to approximate the center of mass.)

By restricting the placement point to coincide with a particular point on the table, named the **target point**, and by restricting the placement plane to be in contact with the table, we can define a family of placements corresponding to this placement class.

With each placement class $\mathcal{P}_j$ we associate a coordinate frame attached to the object and written $\mathbf{P}_j$. The placement frame $\mathbf{P}_j$ is defined as follows: its origin P_j is the placement point, the vector z_{P_j} is the *inward*-pointing normal to the support surface. The orientation of the x- and y-axes is chosen arbitrarily (see Figure 6.6(a)).

[2] The convex hull of a part is the smallest convex object that completely contains the part [69].

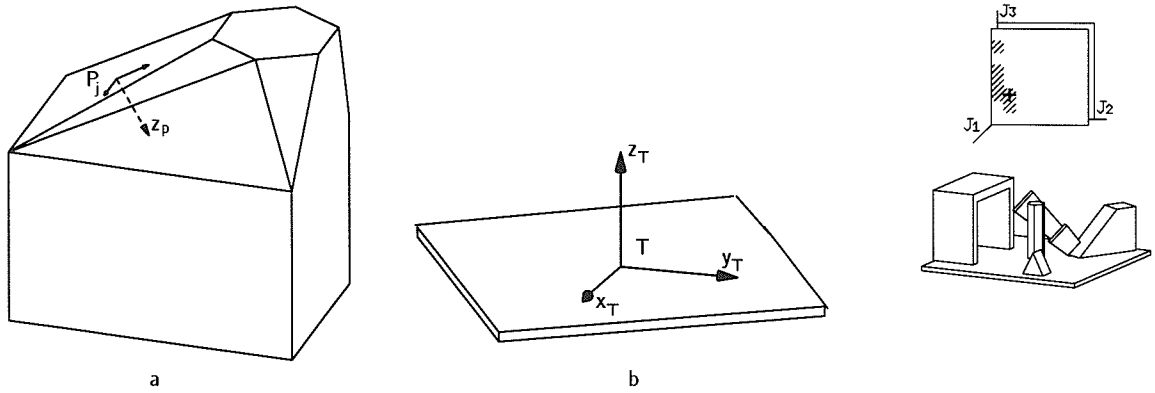


Figure 6.6
(a) Definition of the placement frame. (b) Definition of the target frame

We also define a coordinate frame attached to the table named the **target frame**. The target frame $\mathbf{T}$ is defined as follows: vector z_T is normal to the table. The direction of the x- and y-axes is chosen arbitrarily. The origin of the target frame, T, is located on the table[3] (see Figure 6.6(b)).

By restricting the type of placements, the relative position of the placement frame and the target frame can be parameterized by a single parameter, ϕ, corresponding to the angle between x_T and x_{P_j} (see Figure 6.7). Since the placement frame is attached to the table, the absolute location of the object is totally determined by P_j and ϕ:

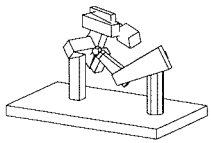
$$\mathbf{T}\mathbf{R}_z(\phi)\mathbf{P}_j^{-1} \tag{6.2.2}$$

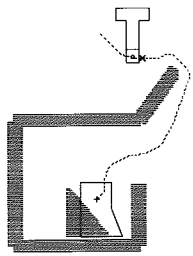
where $\mathbf{P}_j$ denotes the relative position of the placement frame to the object frame and $\mathbf{R}_z(\phi)$ a rotation about the z axis. We denote a particular placement from the placement class $\mathcal{P}_j$ as:

$$(P_j, \phi)$$

Figure 6.8 shows two placements which belong to the same placement class.

[3]The fixed location at which regrasping is accomplished is chosen to minimize collisions with local objects and to have a rich set of nearby kinematic solutions for the robot.





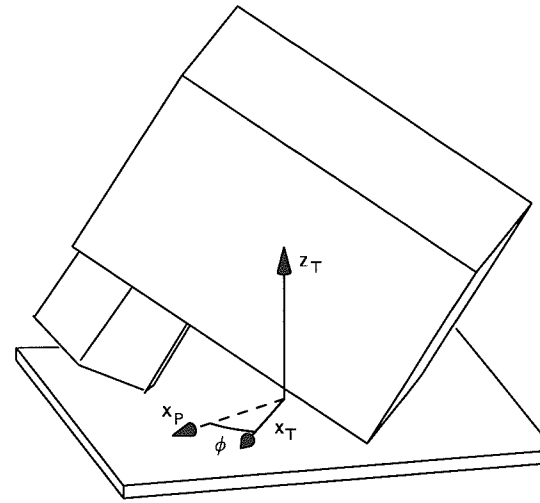


Figure 6.7
Parameterizing a placement $\mathcal{P}_j$ with ϕ

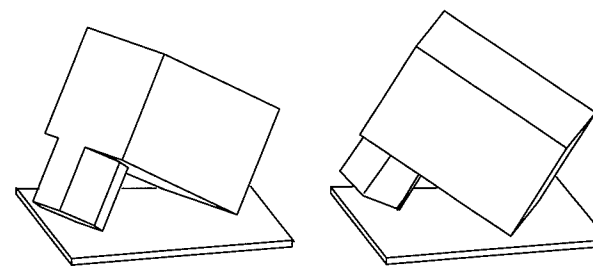


Figure 6.8
Two placements of the same placement class

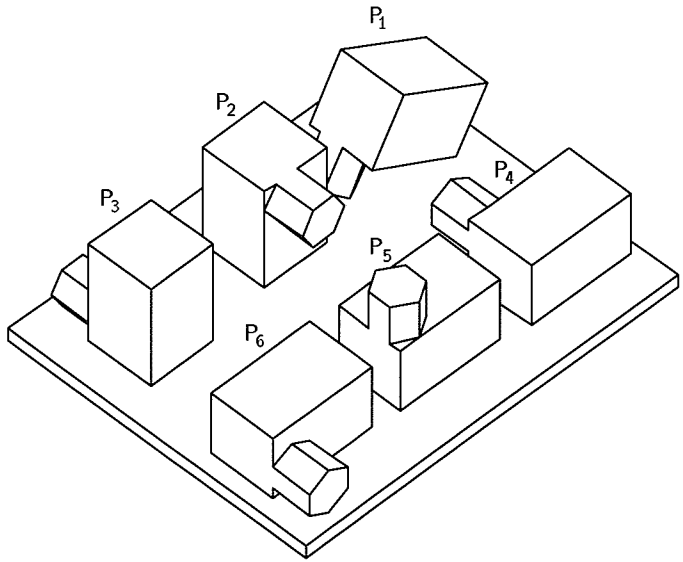


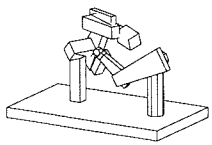
Figure 6.9
All the stable placements of a sample part on the table.

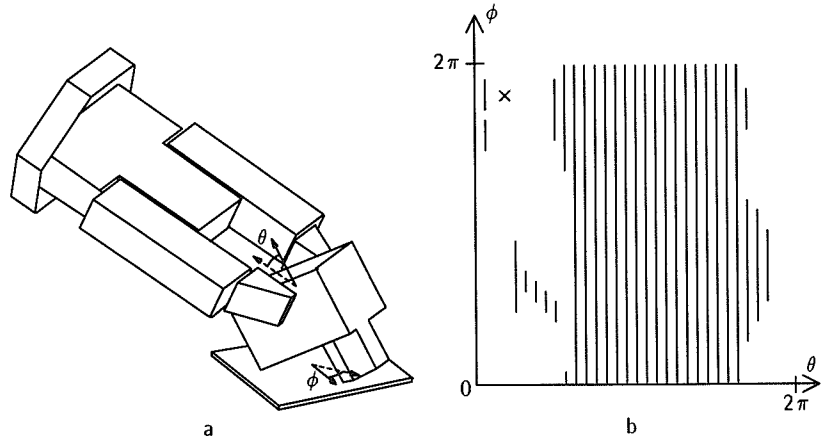
The set of placement classes is obtained by computing the convex hull of the object. The stable faces are selected as well as the placement frames. Figure 6.9 shows all the different placement classes found for the sample part. For each placement class, the part is shown in the $\phi = 0$ orientation.

The choice of the target frame origin, T, should be such that there is a wide range of orientations available to the gripper when the fingers are placed above the target. In principle, the method presented here can easily be extended to handle multiple target frames, but HANDEY does not do this.

6.3 Constructing the legal grasp/placement pairs

We can now define the regrasping problem more formally. The conjunction of a placement (P_j, ϕ) and a grasp (G_i, θ) define how to grasp the part placed at a particular location on the table. The placement defines





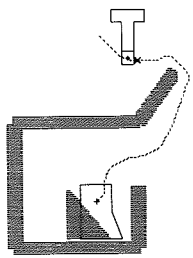


Figure 6.10
Fixing the location of the gripper with (θ, ϕ)

the location of the part relative to the table (6.2.2) and the grasp defines the location of the gripper relative to the part (6.1.1). Together they completely specify the location of the finger frame, which determines the wrist frame. Figure 6.10(a) shows the location of the wrist corresponding a given pair (θ, ϕ). The wrist frame, given a specific grasp and placement, is:

$$\mathbf{T}\mathbf{R}_z(\phi)\mathbf{P}_j^{-1}\mathbf{G}_i\mathbf{R}_z(\theta)\mathbf{F}^{-1}$$

Not all combinations of grasps and placements can be reached by the robot: there must be a set of joint angles for the robot that places the finger frame at the required position and orientation. Moreover, we must take into account constraints of reachability due to:

1. possible collisions of the object and the hand—this restricts legal choices of (G_i, θ) parameters;
2. possible collisions of the hand and the environment at the place where regrasping is performed—this restricts legal choices of conjunctions of (G_i, θ) and $\mathcal{P}_j$; and
3. mechanical bounds on the joint angles—this restricts legal choices of conjunctions of (G_i, θ) and (P_j, ϕ).

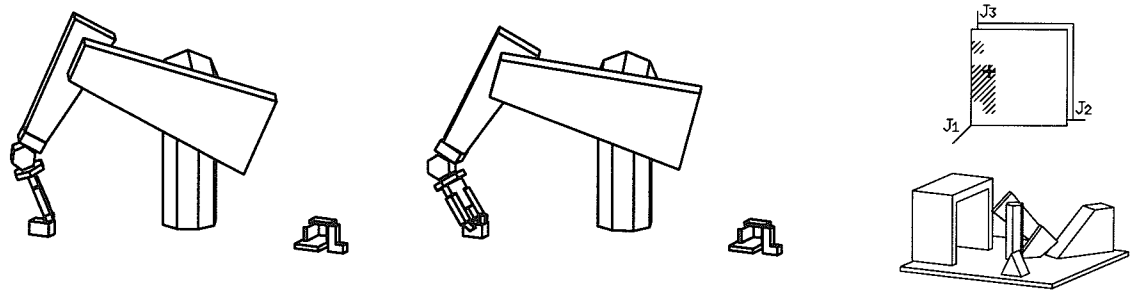
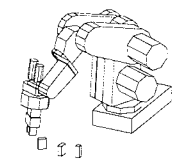
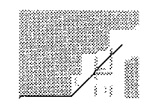


Figure 6.11
A transition between two grasps using a placement (P_j, ϕ).

For each pair $(\mathcal{G}_i, \mathcal{P}_j)$ we compute the set $\mathbf{q}_{i,j}$ of legal (θ, ϕ) by pruning the set $([0, 2\pi), [0, 2\pi))$ using each of the three above constraints (see Section 6.4). A pair $(\mathcal{G}_i, \mathcal{P}_j)$ represents a legal conjunction of a grasp and a placement if this set $\mathbf{q}_{i,j}$ is non-empty. (The $\mathbf{q}_{i,j}$ will be empty if the above contraints prevent the object in the placement class $\mathcal{P}_j$, at any orientation from being grasped.) For example, Figure 6.10(b) shows the subset of legal (θ, ϕ) value for the grasp-placement pair shown in Figure 6.10(a). The dark regions are illegal values of (θ, ϕ). The small '$\times$' indicates the particular (θ, ϕ) values shown in Figure 6.10(a).

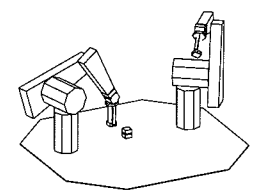


We can now refine the description of regrasping operations: *a regrasping step is a transition between two grasps,* (G_{i_1}, θ_1) *and* (G_{i_2}, θ_2), *taking advantage of a placement* (P_j, ϕ) *compatible with both grasps.* The two statements below give a more formal characterization of those jumps. The second one expresses the duality between grasps and placements.



1. A transition between the grasps (G_{i_1}, θ_1) and (G_{i_2}, θ_2) is legal if and only if there exists a placement (P_j, ϕ) such that:

$$(\theta_1, \phi) \in \mathbf{q}_{i_1,j} \text{ and } (\theta_2, \phi) \in \mathbf{q}_{i_2,j}$$



This simply states that with a grasp of class $\mathcal{G}_{i_2}$, we can pick up an object that has been ungrasped from a grasp of class $\mathcal{G}_{i_1}$. For example, Figure 6.11 shows such a transition.

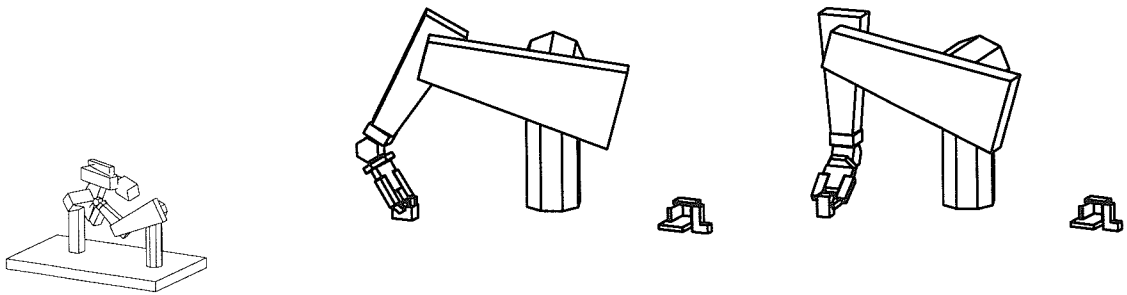
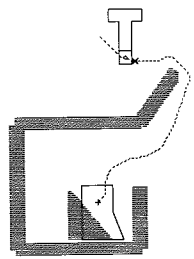


Figure 6.12
A transition between two placements using a grasp (G_i, θ).

2. A transition between the placements (P_{j_1}, ϕ_1) and (P_{j_2}, ϕ_2) is legal if and only if there exists a grasp (G_i, θ) such that:

$$(\theta, \phi_1) \in \mathbf{q}_{i,j_1} \text{ and } (\theta, \phi_2) \in \mathbf{q}_{i,j_2}$$



This states that with a grasp of class $\mathcal{G}_i$, we can move a part between the two placements (P_{j_1}, ϕ_1) and (P_{j_2}, ϕ_2). For example, Figure 6.12 shows such a transition.

One version of the regrasping problem can now be stated: given some initial grasp (G_{i_0}, θ_{i_0}) and some goal grasp (G_{i_g}, θ_{i_g}), find a sequence of grasps $(G_{i_0}, \theta_0), ..., (G_{i_n}, \theta_n)$, such that $(G_{i_n}, \theta_n) = (G_{i_g}, \theta_{i_g})$, and there exists a legal transition between consecutive grasps. If there are multiple solutions to the problem, we want a sequence of minimum length.

The search for a regrasp sequence uses the set defined by

$$\{(\mathcal{G}_i, \mathcal{P}_j) \mid i \in \{1, 2..., n\}, j \in \{1, 2, ..., m\}\}$$

This set can be represented with a **grasp/placement table** (see Figure 6.13). In this table, we use a circle to represent $(\mathcal{G}_i, \mathcal{P}_j)$ pairs with non empty $\mathbf{q}_{i,j}$. The same figure shows the sets $\mathbf{q}_{i,j}$ associated with the $(\mathcal{G}_i, \mathcal{P}_j)$ labeled a, b, c, d and a particular valid configuration for each $\mathbf{q}_{i,j}$. A horizontal edge in this table represents a jump between two placements with a common grasp; a vertical edge represents a jump between two grasps with a common placement. The path shown in the grasp/placement table of the Figure 6.13 corresponds to a two-step regrasp, such as the one shown in Right-Margin Movie 2.

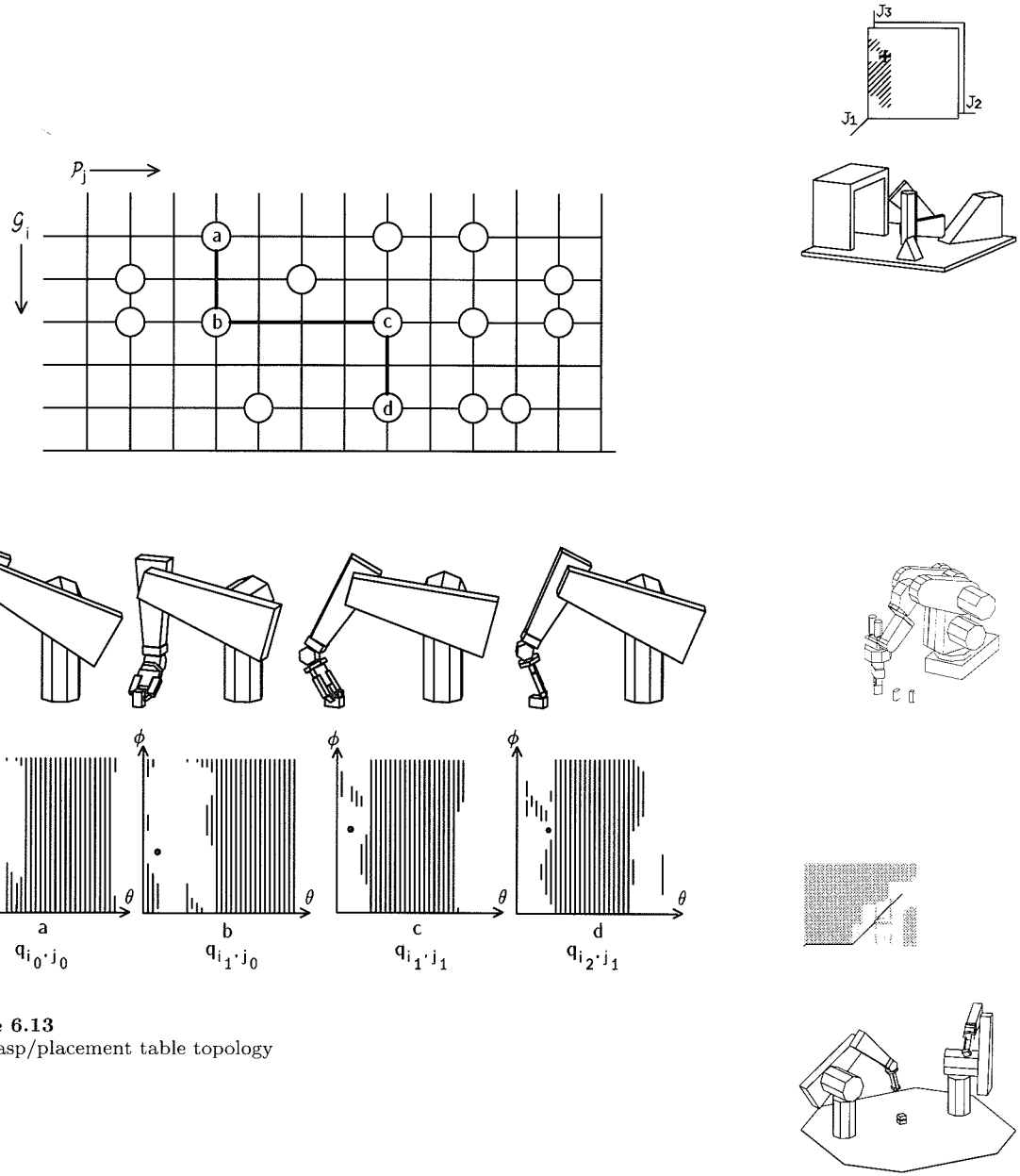


Figure 6.13
The grasp/placement table topology

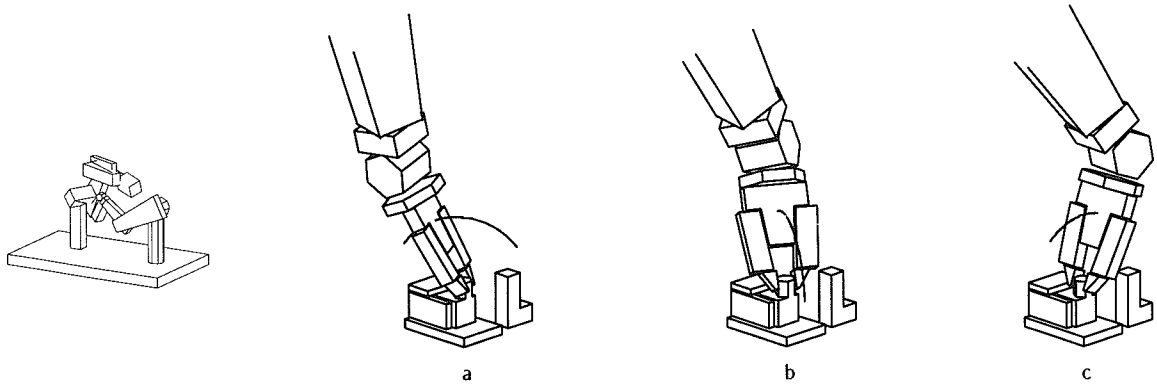
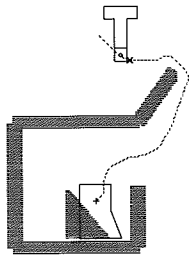


Figure 6.14
All the grasp classes compatible with the putdown pose.



In practice, many versions of the regrasp problem exist. For example, in HANDEY the initial grasp is fully specified by the grasp planner, but the final grasp class $\mathcal{G}_{i_g}$ can be chosen among several possible candidates compatible with the putdown pose (see Figure 6.14). In order not to overconstrain the problem, the grasp angle is not bound to a fixed value for each $\mathcal{G}_{i_g}$ but is only constrained to be in some subset of $[0, 2\pi)$ compatible with the putdown pose. We denote this subset as Θ_{i_g}. Figure 6.15 shows the extreme angles for the grasp class in Figure 6.14(c).

An example of another version of the regrasp problem is to find an initial grasp if the part already lying on the table and the final grasp has been specified. In any case, the main technique for solving regrasp problems is to propagate legal ranges for both placements and grasps along the regrasp sequence. We illustrate this technique in the next section by solving the regrasp problem as it is defined in HANDEY.

6.4 Solving the regrasp problem in HANDEY

HANDEY plans the regrasp sequence after grasp planning is completed and the initial grasp is fully specified. Note that this eliminates a valuable degree of freedom for regrasping. We could, instead, plan the regrasp sequence before the initial grasp in order to minimize the regrasp sequence length. We have chosen not to do this to simplify the imple-

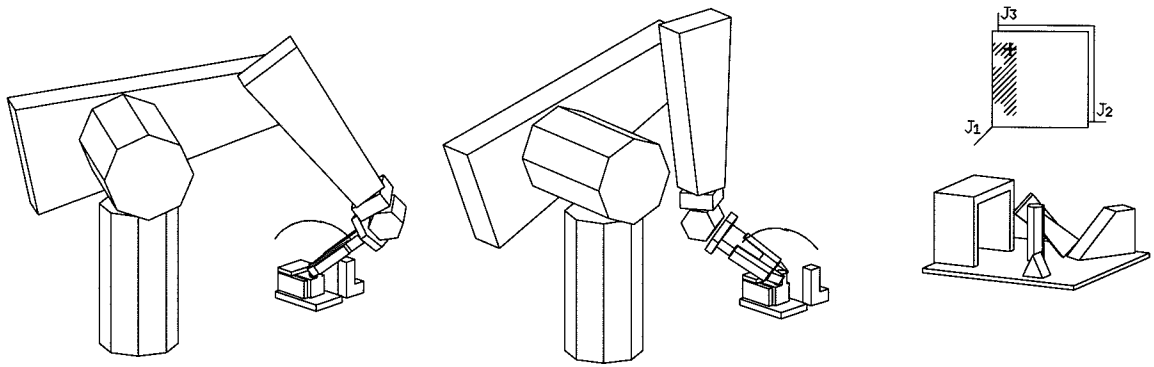
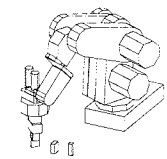


Figure 6.15
The extreme angles for the grasp class in Figure 6.14(a).

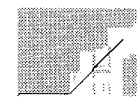
mentation of the grasp planner and the regrasp planner. On the other hand, the regrasp sequence *is* planned before the final putdown grasp is chosen.



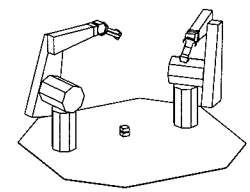
Before searching for a regrasp sequence, the regrasp planner identifies the set of "boundary conditions" for the search, namely,

- the set of initial placements compatible with the initial grasp and the corresponding ranges of the angle ϕ, and
- the set of grasps compatible with the putdown pose and the corresponding ranges of the angle θ.

6.4.1 Computing the initial placements



In HANDEY, the initial grasp typically does not belong to any of the precomputed grasp classes $\mathcal{G}_i$, since it has been computed independently by the grasp planner (see Chapter 5). This does not present a fundamental problem, but it does mean that we have to compute the entry in the grasp/placement table for this initial grasp. This requires computing the legal range of ϕ for each placement. Note that θ is fixed for this grasp.

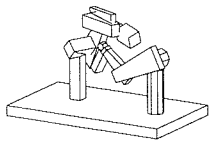


For each placement class $\mathcal{P}_j$, we sample the interval $[0, 2\pi)$ to find the range, Φ_j, which makes this placement compatible with the initial grasp. In practice we take advantage of the local symmetry of the table around the regrasp location to reject immediately any placement (P_j, ϕ)

for which the gripper collides with the table. We sort the set of initial placements by the length of the largest connected subrange of Φ_j.

We define (P_j, Φ_j) as this set of compatible placements as follows:

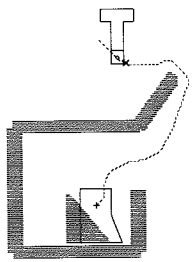
$$(P_j, \Phi_j) = \{(P_j, \phi) | \phi \in \Phi_j\}$$



We call this a **constrained placement**.

6.4.2 Computing the final grasps

Given the putdown pose of the part and a chosen grasp class $\mathcal{G}_i$, we compute the range, Θ_i, that produces grasps compatible with this putdown pose. We could simply sample the interval $[0, 2\pi)$ and check for collisions and inverse kinematic solutions for each value of θ. In practice, we first compute the collision-free rotation range of the gripper around the grasp point with the same algorithm used in the path planner to obtain the range of legal joint angles for a link rotating about a joint axis (see Section 4.2). This range is then sampled and tested for kinematic feasibility. A legal range of θ is illustrated in Figure 6.15.



We sort the set of grasp classes by the length of the largest connected subrange of Θ_i. The grasp class having the largest connected subrange is the most favorable for placing the part at the putdown pose. Figure 6.14 shows all the grasp classes having a non-empty final range for our example.

We define (G_i, Θ_i) as follows:

$$(G_i, \Theta_i) = \{(G_i, \theta) | \theta \in \Theta_i\}$$

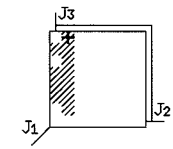
We call this a **constrained grasp**.

6.4.3 Finding a path through the table

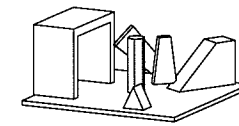
We can now give a more detailed definition of the HANDEY regrasp problem. Given a constrained placement (P_j, Φ_j) and a constrained grasp (G_i, Θ_i) find (θ, ϕ) with $\theta \in \Theta_i$ and $\phi \in \Phi_j$ such that given an initial placement (P_j, ϕ) the grasp (G_i, θ) is achievable after one or more regrasp operations. We call such a problem a **grasp/placement problem** and denote it as

$$((G_i, \Theta_i), (P_j, \Phi_j))$$

A classical breadth-first search is used to find a regrasp sequence that solves such a problem. The definition of the search requires two basic functions:



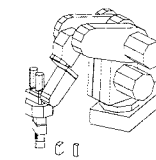
1. an **end-test function** to determine if a given grasp/placement problem $((G_i, \Theta_i), (P_j, \Phi_j))$ leads to an immediate solution to the problem,
2. A **generate function** to produce subgoal grasp/placement problems, that is, problems that if solved will lead to the solution of the current problem.



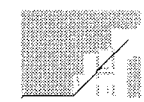
The initial state for the search consists of an initial constrained placement, (P_{j_k}, Φ_{j_k}), denoting a range of initial placements accessible with the current grasp and a constrained goal grasp, (G_{i_g}, Θ_{i_g}), denoting a range of grasps accessible at the putdown pose. A solution to the problem will be found if there is a solution to the grasp/placement problem, $((G_{i_g}, \Theta_{i_g}), (P_{j_k}, \Phi_{j_k}))$.

The generate function will produce subgoals having the form:

$$((G_{i_l}, \Theta_{i_l}), (P_{j_k}, \Phi_{j_k}))$$



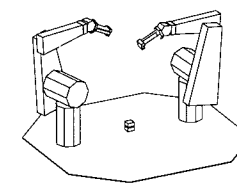
Note that the constrained placement stays unchanged in the definition of each new subgoal. This strategy can be interpreted as backward chaining from the goal grasp. Technically, it would also be possible to proceed forward from the initial placement; in this case we would have generated problems of the form: $((G_{i_g}, \Theta_{i_g}), (P_{j_l}, \Phi_{j_l}))$. But, since there are typically many more grasps than placements, the chance of ending at the desired grasp starting from the initial placement is smaller than the chance of ending at the initial placement starting from the goal grasp.



The end-test function Given a problem $((G_{i_g}, \Theta_{i_g}), (P_{j_k}, \Phi_{j_k}))$ a direct solution exists if:

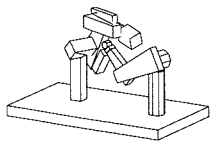
$$\exists\phi_0 \exists\theta_0 \text{ with } \phi_0 \in \Phi_{j_k} \text{ and } \theta_0 \in \Theta_{i_g} \text{ and } (\theta_0, \phi_0) \in \mathbf{q}_{i_g, j_k}$$

That is,



1. We can place the part on the table using the current grasp and placement (P_{j_k}, ϕ_0), since $\phi_0 \in \Phi_{j_k}$.
2. We can grasp (*regrasp*) the part at this location using the grasp (G_{i_g}, θ_0) since the placement (P_{j_k}, ϕ_0) is compatible with this grasp, that is, $(\theta_0, \phi_0) \in \mathbf{q}_{i_g, j_k}$.
3. We can carry the part to its final location, since $\theta_0 \in \Theta_{i_g}$.

Generating subgoals If a solution cannot be found with the current $((G_{i_g}, \Theta_{i_g}), (P_{j_k}, \Phi_{j_k}))$, then we introduce intermediate grasps into the regrasp sequence, by recursively generating new grasp/placement problems. To generate a new problem we proceed in two steps.

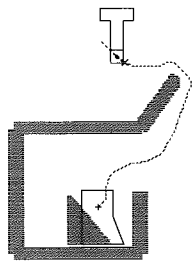


1. We find an intermediate placement class $\mathcal{P}_{j_1}$ compatible with the goal class of grasps $\mathcal{G}_{i_g}$, that is, a placement class for which the following set is not empty (see Figure 6.16):

$$S_{\Theta_{i_g}} = \{(\theta, \phi) | (\theta, \phi) \in \mathbf{q}_{i_g, j_1} \text{ and } \theta \in \Theta_{i_g}\}$$

Further, we compute the following range for ϕ:

$$\Phi_{j_1} = \{\phi | \exists (\theta, \phi) \in S_{\Theta_{i_g}}\}$$



For any $\phi_1 \in \Phi_{j_1}$ the placement (P_{j_1}, ϕ_1) is compatible with the grasp class $\mathcal{G}_{i_g}$ **and** with the range of Θ_{i_g} available for that grasp class. That is, if the part is placed on the table at this placement we can always find a θ_1 in the constrained grasp.
2. Next, we compute a class of grasps and a range that are compatible with (P_{j_1}, Φ_{j_1}). Namely, we find an intermediate class of grasps $\mathcal{G}_{i_1}$ which is compatible with $\mathcal{P}_{j_1}$, in addition we require this class of grasps to be compatible with the range Φ_{j_1}. We compute the set (see Figure 6.17):

$$S_{\Phi_{j_1}} = \{(\theta, \phi) | (\theta, \phi) \in \mathbf{q}_{i_1, j_1} \text{ and } \phi \in \Phi_{j_1}\},$$

and we define a new goal range for θ:

$$\Theta_{i_1} = \{\theta | \exists (\theta, \phi) \in S_{\Phi_{j_1}}\}$$

Consider the new problem defined by $((G_{i_1}, \Theta_{i_1}), (P_{j_k}, \Phi_{j_k}))$ and suppose we get a solution (θ_1, ϕ_1). Given the initial grasp condition, we can place the part on the table using the placement (P_{j_k}, ϕ_1) and, possibly after several regrasps, have the part in the gripper using the grasp (G_{i_1}, θ_1) with $\theta_1 \in \Theta_{i_1}$. At this point it is possible to show that, given the grasp (G_{i_1}, θ_1), we can reach the goal grasp:

1. We can put down the part at the placement (P_{j_1}, ϕ_0) with $\phi_0 \in S_{\Phi_{j_1}}$,
2. Since $\phi_0 \in \Phi_{j_1}$ we can grasp (*regrasp*) the part with (G_{i_g}, θ_0) and $\theta_0 \in \Theta_{i_g}$, that is, a grasp suitable for the final pose of the part.

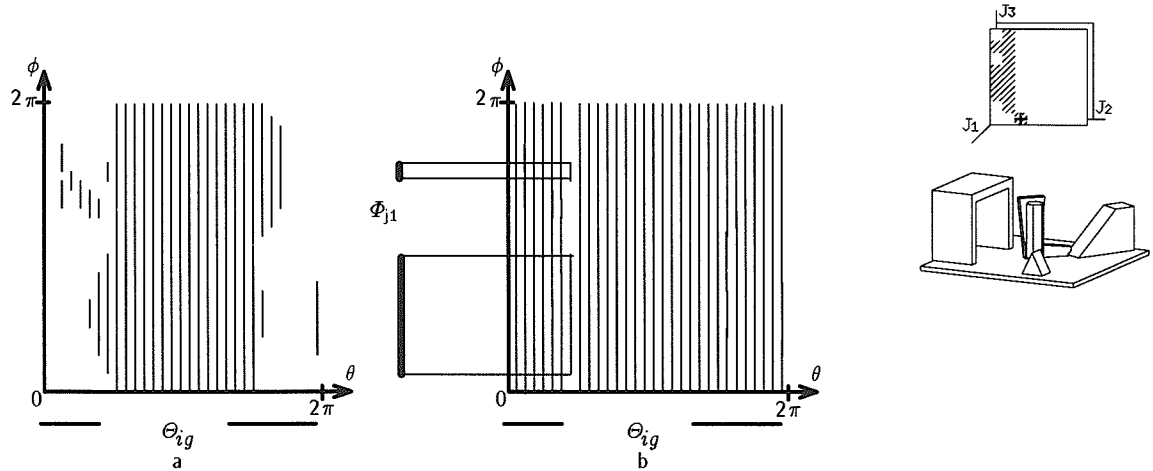


Figure 6.16
Constraining a placement with a constrained grasp

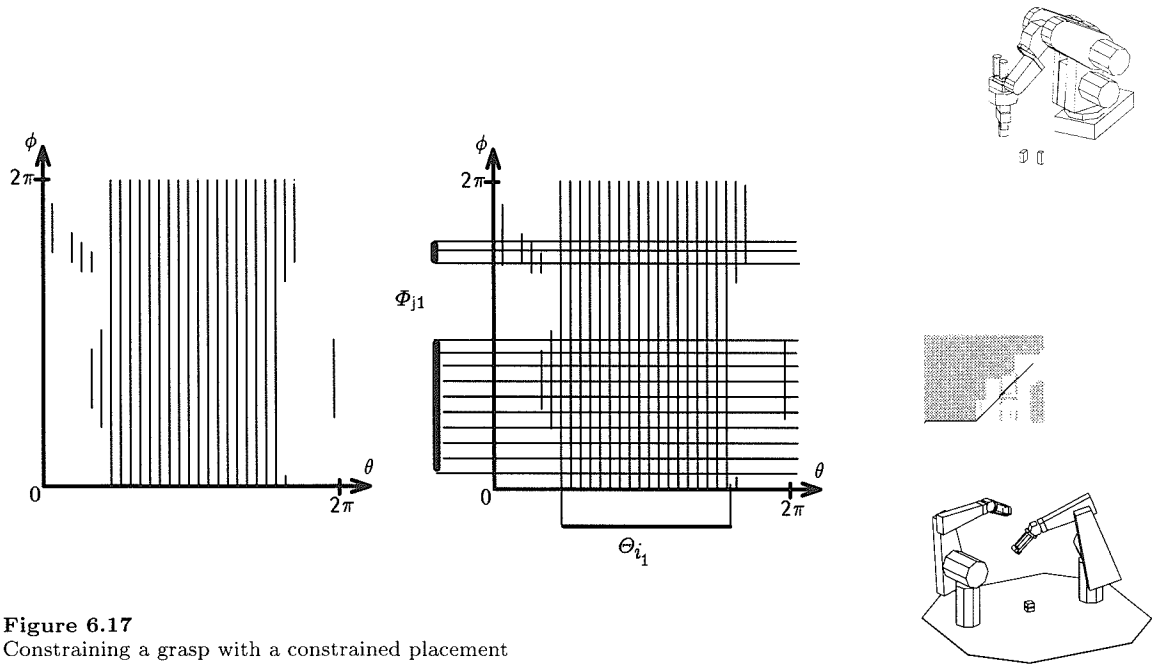
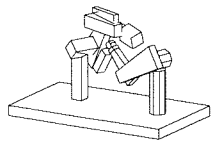
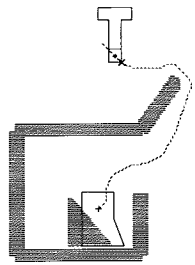


Figure 6.17
Constraining a grasp with a constrained placement



More generally, if we get a solution for a problem at depth n then it is possible to find the solution for the problem which generated this problem at depth $n-1$ and, as a consequence, find a solution for the initial problem (depth 0). Potentially, a problem $((G_{i_g}, \Theta_{i_g}), (P_{j_k}, \Phi_{j_k}))$ can generate several alternative subgoal problems, that is, there are several possible intermediate placements for a given grasp and, again, several possible grasps for a given intermediate placement. To obtain a breadth first search strategy that minimizes the number of regrasping operations, these problems need to be evaluated with the test function before they are expanded into new problems.

6.4.4 Similar problems



During the breadth-first search, it is possible to generate a new problem having an ancestor sharing the same grasp class but with a different constraint on θ. We call such a problem a **similar problem**. Similar problems need to be treated specially to avoid infinite recursion.

When generating new problems, the system maintains a table of all the previously generated problems and, more particularly, the list

$$(G_i, \bigcup_{l=1}^{q} \Theta_i^l)_{i \in \{1,2,\ldots,n\}}$$

where $\bigcup_{l=1}^{q} \Theta_i^l$ denotes the union of the constraints previously generated for the grasp class $\mathcal{G}_i$. When a similar problem is generated, $((G_i, \Theta_i^{q+1}), (P_{j_k}, \Phi_{j_k}))$, the system replaces it with the problem:

$$((G_i, \Theta_i^{q+1} - \bigcup_{l=1}^{q} \Theta_i^l), (P_{j_k}, \Phi_{j_k}))$$

If the set $\Theta_i^{q+1} - \bigcup_{l=1}^{q} \Theta_i^l$ is empty, it does not expand the problem further. This reduction is made with no loss of generality because of the following two properties:

1. If the problem $((G_i, \Theta_{i_1}), (P_{j_k}, \Phi_{j_k}))$ has a solution and if $\Theta_{i_1} \subseteq \Theta_{i_2}$, then it is the case that $((G_i, \Theta_{i_2}), (P_{j_k}, \Phi_{j_k}))$ has a solution.

According to our definition of a problem, the solution for the first problem is also a solution for the second one. As a consequence we get the second property:

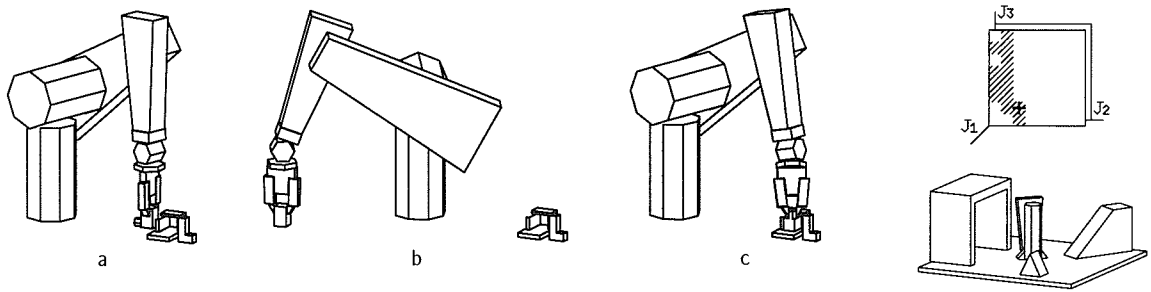
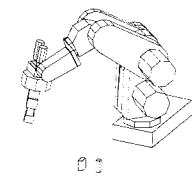


Figure 6.18
Geometric boundary conditions for the regrasp problem in HANDEY: (a) the initial grasp of the part; (b) the initial placement in the regrasp sequence; (c) the goal, or putdown, grasp.

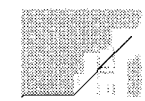
2. The problem $((G_i, \Theta_{i_1} \cup \Theta_{i_2}), (P_{j_k}, \Phi_{j_k}))$ has a solution if and only if either the problem $((G_i, \Theta_{i_1}), (P_{j_k}, \Phi_{j_k}))$ has a solution or the problem $((G_i, \Theta_{i_2}), (P_{j_k}, \Phi_{j_k}))$ has a solution.

So, by replacing the problem $((G_i, \Theta_i^{q+1}), (P_{j_k}, \Phi_{j_k}))$ with the problem $((G_i, \Theta_i^{q+1} - \bigcup_{l=1}^{q} \Theta_i^l), (P_{j_k}, \Phi_{j_k}))$, we consider only the part of the problem which is new to us. We assume that if there is a solution for the previously generated ranges it will be found by expanding another branch of the tree.

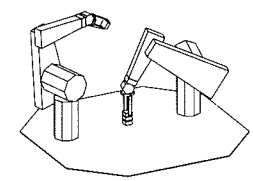


6.5 An example

In this section we work through a particular example to illustrate the algorithm described above. The initial grasp of the part is given. The initial placement and the putdown grasp are computed with their respective legal angle ranges (see Figure 6.18). That defines the initial problem: $((G_{i_g}, \Theta_{i_g}), (P_{j_k}, \Phi_{j_k}))$.



The end-test function is applied to this initial problem. In this particular example the set $\mathbf{q}_{i_g,j_k}$ is empty: the goal grasp is not compatible with the initial placement for any value of (θ, ϕ). At this point, a single-step regrasp operation is impossible given the initial placement and goal grasp—we need to expand the problem.



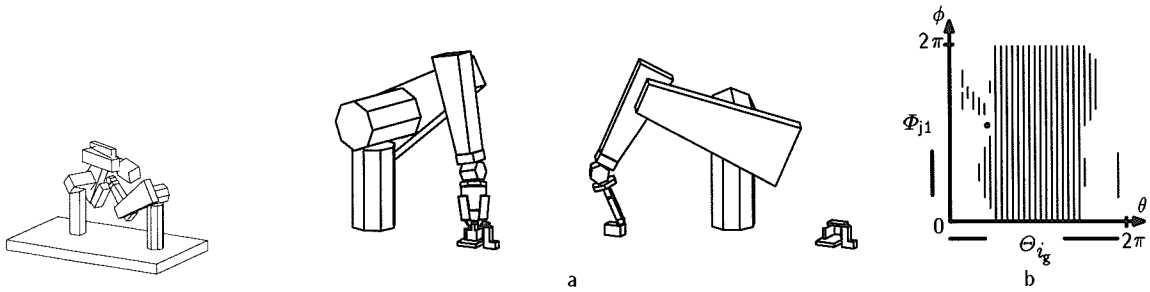


Figure 6.19
Intermediate placement compatible with goal grasp.

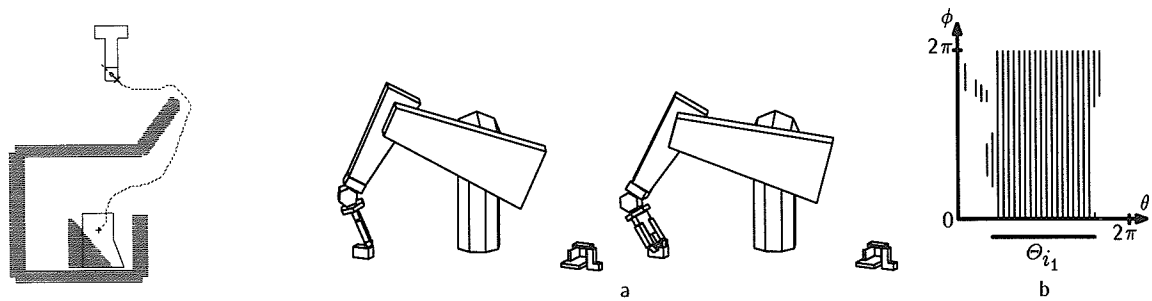


Figure 6.20
Intermediate grasp compatible with intermediate placement.

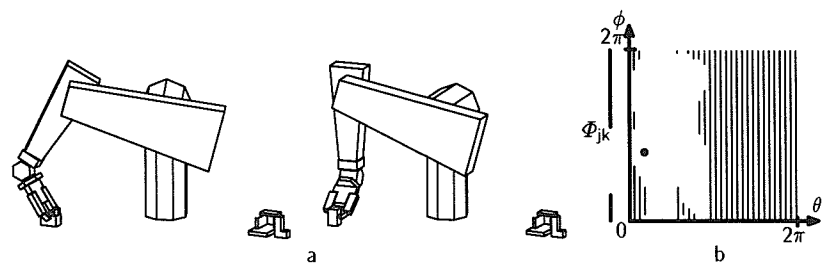
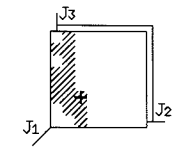
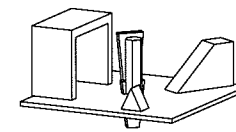


Figure 6.21
Intermediate grasp compatible with initial placement.

- **Step 1**: A placement class $\mathcal{P}_{j_1}$ compatible with $\mathcal{G}_{i_g}$ is found. The set $\mathbf{q}_{i_g,j_1}$ and the constraints on the legal values for θ, Θ_{i_g}, are used to compute the constraints on this placement Φ_{j_1} (see Figure 6.19).
- **Step 2**: Given (P_{j_1}, Φ_{j_1}) a compatible grasp class $\mathcal{G}_{i_1}$ is found. The set $\mathbf{q}_{i_1,j_1}$ and the constraints on the legal values for ϕ, Φ_{j_1}, are used to compute the constraints on this grasp Θ_{i_1} (see Figure 6.20).

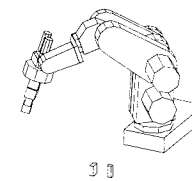




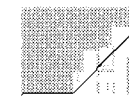
Finally we consider the problem $((G_{i_1}, \Theta_{i_1}), (P_{j_k}, \Phi_{j_k}))$. The grasp class $\mathcal{G}_{i_1}$ is compatible with the placement class $\mathcal{P}_{j_k}$ and it is possible to find a solution (θ_1, ϕ_1) in $\mathbf{q}_{i_1,j_k}$ which satisfies the constraints on the ranges for θ and ϕ (see Figure 6.21). The pose corresponding to this solution is shown in Figure 6.18(a). By construction, we can deduce (θ_1, ϕ_0) and (θ_0, ϕ_0) which gives us the desired regrasp plan.

6.6 Computing the constraints

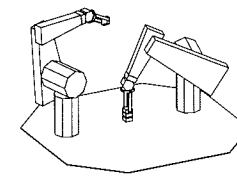
We will now consider in more detail how to compute $\mathbf{q}_{i,j}$, the description of the legal combinations of (G_i, θ) and (P_j, ϕ).



For a given choice of a grasp class $\mathcal{G}_i$ (which implies a choice of the grasp point), the position of the gripper relative to the grasp frame is parameterized by the angle θ. We want to compute the range of θ for which there are no collisions between the gripper and the grasped object, the table, and any nearby obstacles. Note that this problem involves computing the forbidden ranges of angle for a polyhedron (the gripper) rotating about a fixed axis. This is precisely the basic computation used by the slice-projection motion planner described in Chapter 4. This computation must be repeated for each value of ϕ.

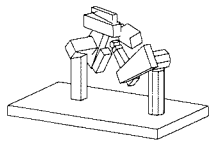


The other constraint that a legal (θ, ϕ) value must satisfy is the existence of at least one solution for the inverse kinematics of the arm for that gripper configuration. In HANDEY this constraint is enforced by sampling the range of collision-free (θ, ϕ) and testing the inverse kinematics for that configuration of the gripper. Sampling in this way can be time consuming since it must be done over a two-dimensional space.



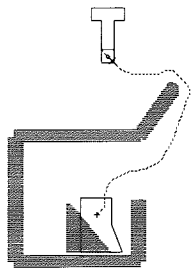
In addition to testing for potential gripper collisions and the kinematic feasibility of the gripper pose, we must also check for potential arm collisions. This is precisely the computations performed by the grasp planner (see Chapter 5) in computing the grasp plane C-space maps.

Computing all the valid grasp/placement constraints by this method is very time consuming. Much of the time is spent on checking kinematic feasibility and computing collisions with fixed objects. In practice, this can be precomputed for the part of the workspace where regrasps are to be carried out.



6.7 Regrasping using two parallel-jaw grippers

We mentioned earlier that instead of using a table to assist during regrasping, one can use another gripper. We will call this second gripper the **left gripper**. There are two simplified versions of this problem that can be treated very similarly to the single gripper with a table case that we treated above. HANDEY does not currently implement these extensions, but they are described here since they are closely related to the earlier development.



6.7.1 Regrasping using a fixed left gripper

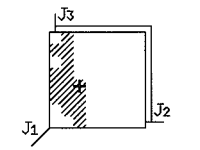
Using the table to support the part during regrasping severely limits the available range of grasps on the part. The presence of the table constrains the legal range of θ for most grasps to be substantially less than π. Consider replacing the table by a left gripper constrained so that it is fixed in space. This gripper can be used to hold the part while the other gripper changes the grasp. This situation is essentially identical to the table case except that the legal ranges of θ can be expected to be larger.

Grasps for the left gripper can be characterized by some grasp class $\mathcal{G}_j$ and rotation angle θ_j. The grasp (G_j, θ_j) plays the role that the placement (P_j, ϕ) played in the previous development. The planning process for this restricted case is exactly analogous.

What have we gained from using another gripper? Two things:

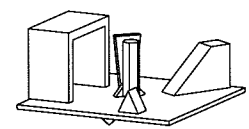
1. The legal (θ, ϕ) for a given grasp/placement is likely to be larger since we have removed the table as an obstacle.
2. There are more available placements since we have removed the stability requirement for placements. Any legal grasp for the left gripper becomes a placement.

As a result, the search space is much larger and less constrained. This increases the chances of finding a one-step regrasp operation.

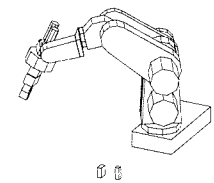


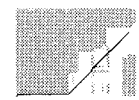
6.7.2 Regrasping using a mobile left gripper

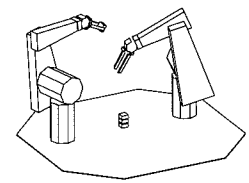
Consider first a simple extension to regrasping with a table: assume that the target point for regrasping lies at the center of a turntable. Then, we can modify the search for a regrasp sequence by dropping the propagation of the ϕ range. If two grasps are compatible with a given placement for any values of ϕ, then we can move from one grasp to the other at that placement simply by using the turntable to select the valid ϕ for each grasp. By introducing a turntable in the system, we drastically increase the number of pairs of grasps which share a common placement. As a consequence, we increase the chances of performing a regrasp operation in one step.

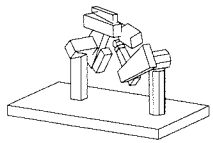


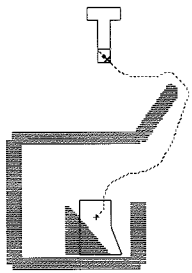
We can now combine the advantages of the left gripper and turntable methods by allowing the left gripper to rotate along its x_F axis. All that is required is to constrain the left gripper so that its reference point F is fixed in space while the gripper is allowed to rotate about x_F. Most robot grippers have a full range of rotation about their gripper axis and, therefore, can readily emulate a turntable.





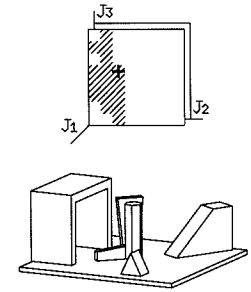






7 Coordinating Multiple Robots

In Chapter 4, we discussed the problem of generating paths for robots while avoiding fixed obstacles in the workspace. Finding paths becomes trickier when some of the obstacles may be moving. In particular, when two robots are operating in close proximity to each other, they each become obstacles to the other. Unfortunately, the configuration space methods described in Chapter 4 are designed to work only for fixed obstacles. In this chapter, we describe a method of coordinating the paths of multiple robots so as to avoid collisions between them (as well as collisions with fixed obstacles).

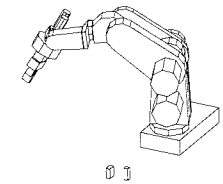


The multi-arm coordinator is invoked as follows:

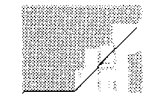
Coordinate$(plan_1, plan_2, world)$

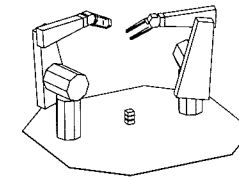
where $plan_1$ and $plan_2$ are plans for two different arms. The multi-arm coordinator produces a combined plan for the two arms that will move the arms through the same paths but allows them to move at the same time (in parallel) whenever possible.

7.1 Coordination and parallelism



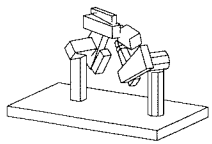
Consider a task where two robots are cooperating in assembling some device from its components. For much of that task, the robots will be working in separate areas of the workspace, and could operate simultaneously. Portions of the task, however, will require the robots to move into some common area of the workspace, where the device is actually being assembled. Since the two robots cannot occupy the same volume at the same time, these portions of the task must be synchronized to ensure that only one robot enters that common portion of the workspace at a time. (For the most part, this chapter deals with two robots, and the examples shown will use just two robots. Please note, however, that much of the development described here is applicable to more than two robots. Discussion of the extensions is deferred to Section 7.6.7.)





The primary goal of the multi-robot planner is to synchronize the paths of multiple robots to avoid collision. We will see that the planner attempts to preserve as much parallelism as it can—allowing all robots to move independently, provided they stay out of each other's way. The multi-robot planner computes a set of synchronizing instructions, specifying which portions of the robots' paths may not be executed simultaneously. We will see that this modular approach provides great

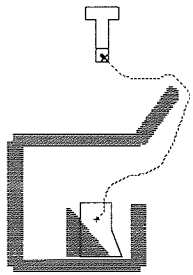
flexibility in the path planner and the actual robot controller for *loosely-coupled* robots. The multi-robot planner does not, however, solve certain types of multiple robot coordination problems, such as *tightly-coupled*, or *dynamically-linked* robots.



We will first take a look at different types of multiple robot coordination, then we will give a short example of the operation of the multi-robot planner. Following the example, we will examine the planner in detail.

7.1.1 Planning motions of more than one robot

There are many possible approaches to multiple robot coordination, though each approach solves a slightly different problem. *Tightly-coupled* robots would include two robots which together manipulate a single part or assembly simultaneously. This would require a precise planning *and control* of both robots' trajectories to prevent unacceptable forces being applied to the assembly. The multi-robot planner does not attempt such precise planning.



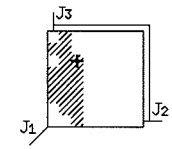
Workspaces that include moving parts as well as robots require a planner to track the parts and to plan the robots' trajectories to avoid those parts as well as the other robots. The multi-robot planner will only synchronize the trajectories of moving objects which can be controlled (the robots).

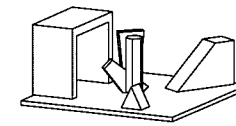
Loosely-coupled robots work together in the same workspace, but do not manipulate the same part at the same time (except to hand-off a part from one robot to another). This is the domain in which the multi-robot planner works.

7.1.2 Goals of the multi-robot planner

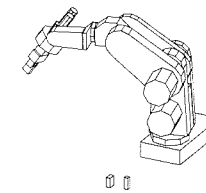
First, a word on terminology. The other planners in HANDEY generate **paths**, which describe the shape of a motion, that is, the shape of the curve in the robot's configuration space. A path does not specify the velocity or acceleration of the robot along that curve. We define a **trajectory** of a robot as the time history of positions along a path, that is, a curve through the robot's **state space**. There are infinitely many trajectories possible for a given path, each differing in the time history of velocities along the path. A path describes through which points in space the robot passes; a trajectory describes when it passes through them.

One key difference between the method used by the multi-robot planner and previous coordination methods is that this method attempts to decouple the path specification step from the trajectory specification step. In particular, the multi-robot planner does not plan paths—that is left to a separate path planning module (see Chapter 4). All collisions are avoided by using *time*, that is, by waiting for the other robot to get out of the way, without changing the path. (We will see, however, that this method also leads to a prescription of how to change paths so as to minimize the interference between the trajectories.) The multi-robot planner does not actually plan trajectories, either, in that we don't control the actual velocities of the robots. We assume that we *can* control the sequencing of the paths of the robots.



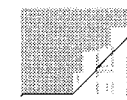


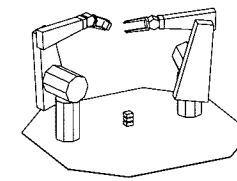
We will be particularly interested in the problem of coordinating the trajectories of robot manipulators working in known, predictable environments. We assume that the paths of the manipulators can be planned off-line to avoid collisions with all the objects in the environment, except the other robot. There are no unforeseen obstacles. This is true of most industrial tasks. We also assume that, although the robots' paths are predictable, their trajectories are less predictable. One example of this is arc welding where the speed may be adjusted in response to observed weld parameters. Another example is when one of the steps in the task may involve a sensor-based operation of varying duration. There are other, simpler reasons for unpredictable trajectories, such as unavoidable error in the controller.

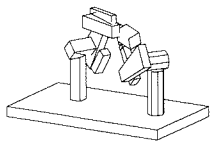


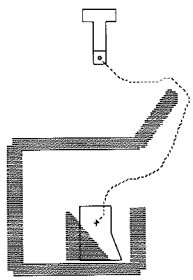
Given the preceeding assumptions, we have the following goals for the multi-robot planner:

- it should be possible to plan the path for each manipulator essentially independently,
- the resulting trajectories should guarantee that the manipulators will reach their goals,
- it should be possible to execute the trajectories without precise time coordination between the manipulators, and
- the safety of the manipulators should not depend on accurate trajectory control of individual manipulators.









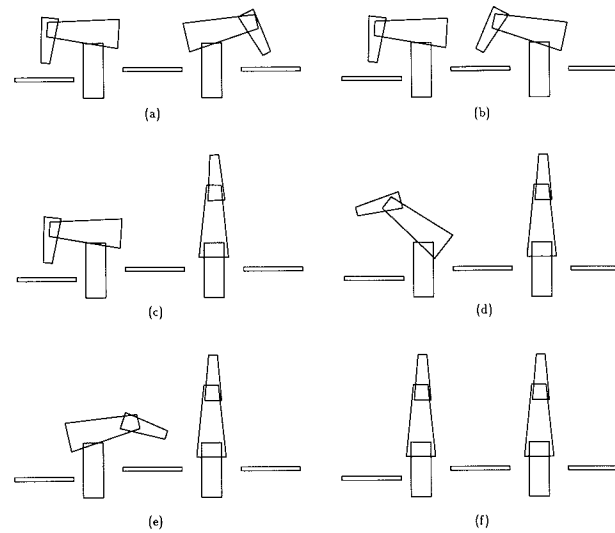


Figure 7.1
Two planar robots working in a common workspace. Simple paths have been planned for these robots to exemplify the multi-robot planner's operation. (No grippers or manipulated parts are drawn to keep the pictures simple.) Note that, although the robots' paths are shown in sequence, the two paths were planned independently, without regard to the position of the other robot.

7.2 Robot coordination as a scheduling problem

The multi-robot planner accepts as input a path for each robot. These paths are specified as a sequence of joint angle vectors, or setpoints. (Other robot commands such as to open and close a gripper may be included. We will have more to say about those in Section 7.4.4.) The output of the multi-robot planner is a **schedule**. A schedule describes the order in which the setpoints for each robot may be sent to their respective controllers for execution. In particular, the schedule specifies where each robot must wait before entering a portion of space that is already occupied by the other robot. The following example illustrates the procedure.

Figure 7.1 shows two planar robots. Paths have been planned, independently, for both robots. Each robot moves from a position above its private work table to the common work table, then to a parked position

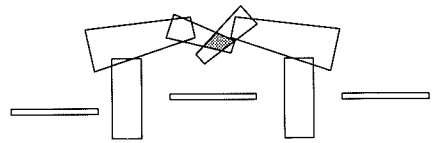


Figure 7.2
A collision would occur if both robots attempted to move to the central work table simultaneously.

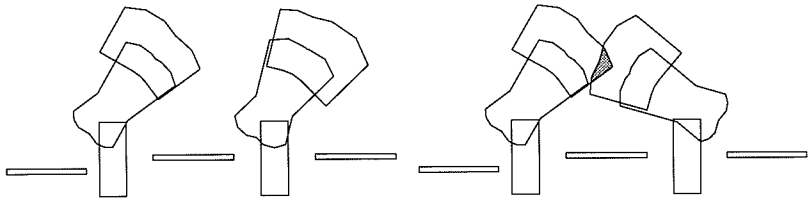
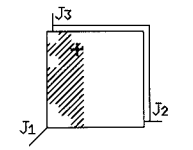
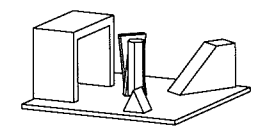


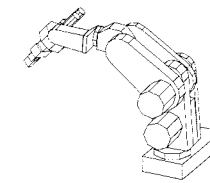
Figure 7.3
(a) Volumes swept out by each robot executing one path segment. (b) Intersecting swept volumes indicating a potential collision. The highlighted area indicates the region of the potential collision.

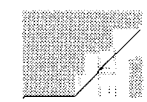
pointing straight up. Note that both robots want to occupy a portion of the region above the common work table for some portion of their respective paths. Naturally, if they were to both move there at the same time, they would collide (see Figure 7.2).

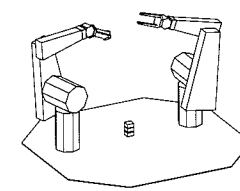
Given the paths for the two robots, we can characterize where the possibility of collision exists. Between each pair of adjacent setpoints on a robot's path the robot sweeps out a particular region of space. We call the motion between adjacent pairs of setpoints **path segments**. By comparing a path segment of the first robot's path with each segment from the second robot's path, we can identify where collisions might occur (see Figure 7.3). If the the same volume of space could be occupied by both robots during the execution of a pair of path segments, we say that the pair of segments creates a **potential collision**. The two robots would likely collide if they were to both execute those path segments simultaneously. Detecting whether two path segments may collide is straightforward (see Section 7.4.6).

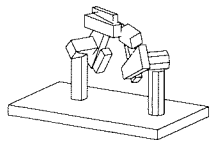












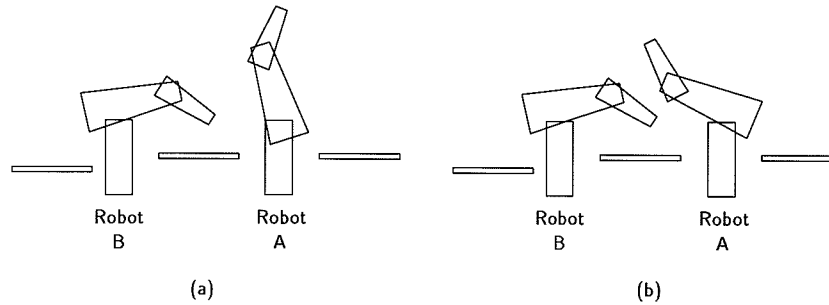
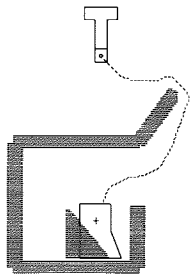


Figure 7.4
(a) Robot A waits for robot B to get out of the way before proceeding along its planned path. (b) Robot A waits for robot B to get out of the way, but, unfortunately, robot B cannot get out of the way because robot A is in the way of robot B's continuation along its planned path.

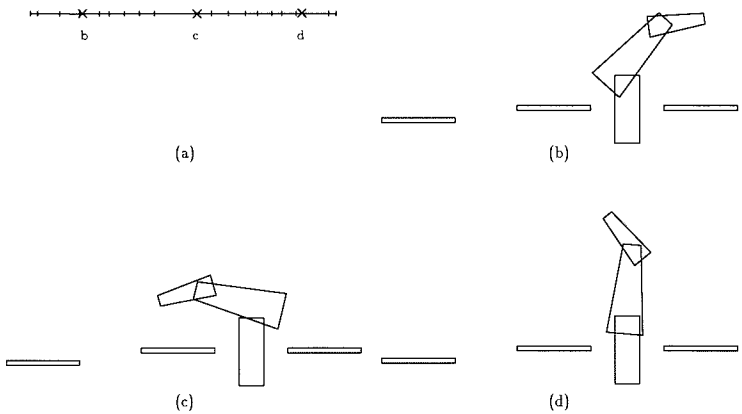


Knowing which path segments could cause a collision between the two robots, it is a simple matter to prevent any collisions by ensuring that segments from each robot which potentially collide are not executed at the same time. Specifically, if robot B is executing a path segment which may collide with the segment that robot A is about to execute, we instruct robot A to wait before executing the next segment until robot B has finished executing segments which may collide with that one (see Figure 7.4(a)). This is the fundamental task which the multi-robot planner performs: identify which path segments might cause collisions and insert wait commands in the paths to prevent simultaneous execution of those pairs of path segments with potential collisions.

7.2.1 Deadlock

Of course, the procedure described so far does not guarantee that the robots will complete their respective tasks. In any scheduling problem involving waiting for resources, the risk of **deadlock** must be addressed. In our case, a deadlock can occur when each robot is waiting for the other robot to complete some path segment (see Figure 7.4(b)). Once a deadlock situation has occured, the task cannot be completed, since neither robot can proceed without danger of collision.

The deadlock situation in Figure 7.4(b) could have been avoided if robot A had stopped earlier in its path to wait for robot B to get out of



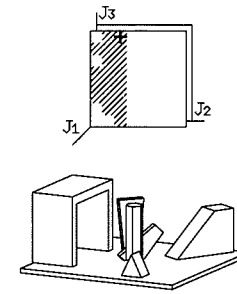


Figure 7.5
(a) The path of one robot represented by a straight line. The hash marks along the line represent the setpoints computed by the path planner. The line segments between the hash marks represent path segments. The line is parameterized by progress along the path, not by time or position. (b)-(d) Positions of the robot as selected by the crosses along the line in (a).

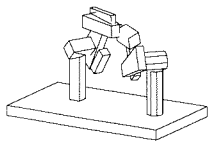
the way, such as in Figure 7.4(a). The trick is to determine how much earlier in robot A's path it has to stop.

7.3 The task completion diagram

We can develop a graphical depiction of the path segments and their potential interaction. This graph will enable us to visualize the scheduling problem before us and immediately suggest a means for avoiding deadlock situations.

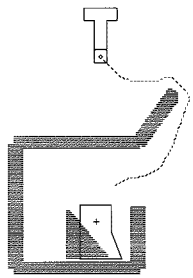
Imagine the path of robot A represented by a straight line. The points along the line represent the configurations of the robot as it progresses along the path. In Figure 7.5(a), the dots along the line are the setpoints determined by the path planner; the line segments between the setpoints represent the path segments we've been talking about. As the robot proceeds along its path, we can further imagine a parameter, similar to a computer's program counter, which increases along this line. Note that distance along the line does *not* represent time or duration in any meaningful way, since the line represents the *path*, not the *trajectory*, as

we have defined those terms earlier. Neither does the parameter directly represent position of the robot in either Cartesian or configuration space, except with reference to the path that the path planner produced.



Let a be the parameter which indicates how far along its path robot A is in its execution of the path. The line becomes a coordinate axis, with a as it's variable. Then a path defines a function $\mathbf{q} = \mathbf{A}(a)$ which specifies the robot's configuration, $\mathbf{q}$, as a function of this parameter. A trajectory defines a function $a = T(t)$ which specifies the path parameter, a, in terms of time, t. The setpoints, numbered 0 through m, are represented by path parameter values $a_0, \ldots, a_m$. We denote a path segment by A_i, which is the portion of the path between the parameter values a_i and a_{i+1}, or, formally,

$$A_i = \{\mathbf{q} = \mathbf{A}(a) \mid a_i \leq a \leq a_{i+1}\}. \tag{7.3.1}$$



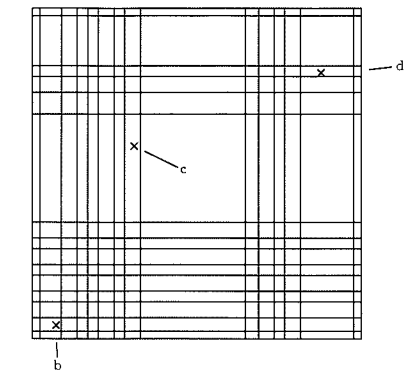
Note that both limiting joint configurations are included in the segment. This has important consequences, as we will see later. We denote the set of points in physical space occupied by the robot as it sweeps along the path segment by $SV(A_i)$.

The distance between setpoints, $a_{i+1} - a_i$, on the "path line" is *not* meaningful, and is arbitrary. (We will see later that we can use distance to encode assumptions about the time duration of segments, and use that in further analysis, but such encoding has limited relation to reality. In the figures in this chapter, we use the distance between setpoints to represent the "size" of the motion in the segment—it is the maximum angle any of the joints must move in that segment. We must emphasize that this is a convenience, and does not affect the multi-robot planner's algorithm in any way.)

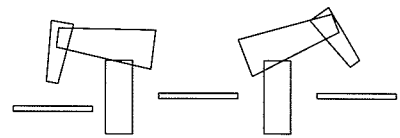
We next place two of these lines at right angles to make a coordinate system—one axis corresponds to each robot. We form a grid using the setpoints; each grid rectangle, which we denote by $R_{i,j}$, represents a pair of path segments, A_i and B_j, one from each robot's path. A point, (a, b), within the diagram represents a configuration, $(\mathbf{q}_A, \mathbf{q}_B)$, of the two robots, where each robot is in the configuration corresponding to the coordinates of the point on the two axes: $\mathbf{q}_A = \mathbf{A}(a)$, $\mathbf{q}_B = \mathbf{B}(b)$ (see Figure 7.6). We can define $R_{i,j}$ formally as:

$$R_{i,j} = \{(\mathbf{q}_A, \mathbf{q}_B) \mid \mathbf{q}_A \in A_i \text{ and } \mathbf{q}_B \in B_j\} \tag{7.3.2}$$

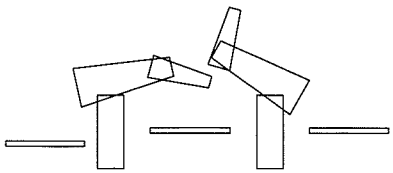
We call this diagram a **task completion diagram**, or TC diagram.



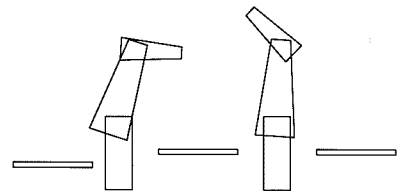
(a)



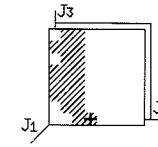
(b)

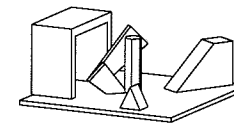


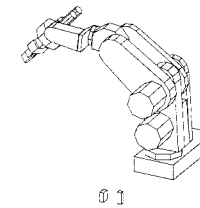
(c)

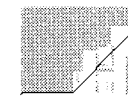


(d)









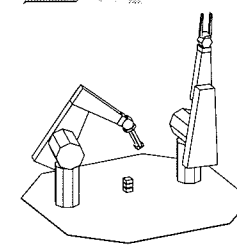
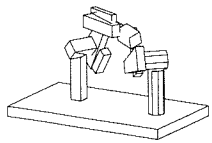
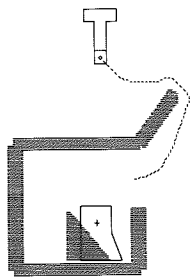


Figure 7.6
(a) The task completion diagram corresponding to our example. The axes are "straight line representations" of the paths of each of the two robots. Each grid rectangle corresponds to a pair of path segments, one from each robot. (b)-(d) Positions of the two robots as selected by the crosses in the TC diagram. Robot A is on the right; robot B is on the left.





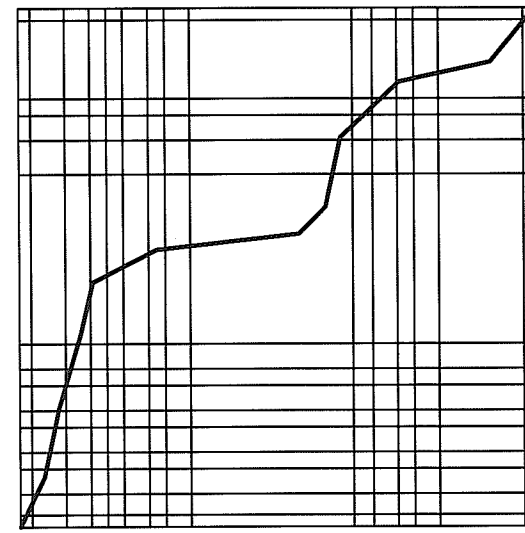
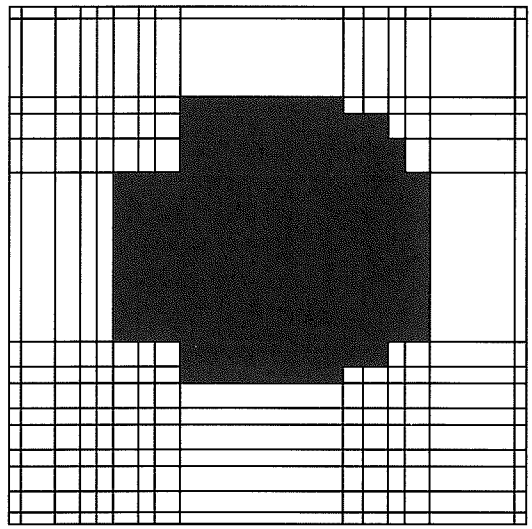


Figure 7.7
A schedule in a TC diagram. The curve (monotonically increasing in both coordinates) represents a sequence of configurations taken by the two robots.)

7.3.1 Schedules

As each robot proceeds along its planned path, its coordinate in the TC diagram monotonically increases. (As we noted above, the coordinate has no physical meaning other than an indication of how far along the path the robot is.) The coordinate pairs generated by the particular trajectories taken by the robots trace out a locus in the TC diagram (see Figure 7.7). We call this curve in the TC diagram a **schedule**. The schedule defines two functions relating the path coordinates for the two robots: $a = f(s)$, and $b = g(s)$, $0 \leq s \leq 1$. Since both robots begin at their starting setpoints and end at their goal setpoints, the two ends of a **completed schedule** must be at the lower left and the upper right corners: $a_0 = f(0)$, $b_0 = g(0)$, $a_m = f(1)$, and $b_n = g(1)$. The schedule curve must always begin at the lower left corner and proceed up and to the right; that is, we do not allow the robots to back up: $f'(s) \geq 0$ and $g'(s) \geq 0$ for all s. (See Section 7.6.6 for discussion of relaxing this requirement.)



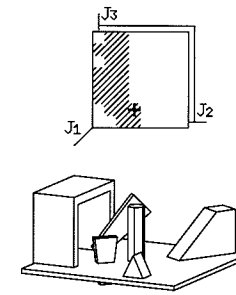
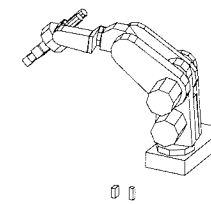
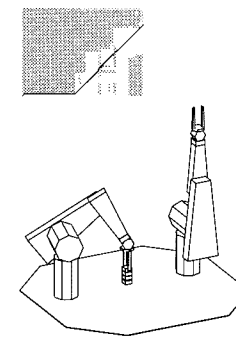


Figure 7.8
The "colored" TC diagram for our sample paths. The shaded grid rectangles represent forbidden regions of the diagram through which a safe schedule must not pass.



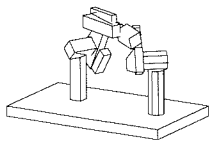
It is plain that there are an infinity of possible schedules, depending on the particular sequence of configurations the robots pass through. Obviously, not all schedules are **safe**, since the robots will collide. Our next step is to characterize all **safe schedules**.

7.3.2 Collision regions

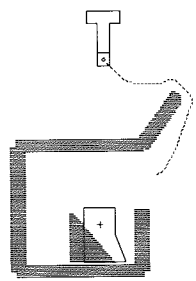


A pair of path segments A_i and B_j will potentially collide if the robots would occupy the same physical space during the segments, $SV(A_i) \cap SV(B_j) \neq \emptyset$. We "color" the corresponding TC diagram grid rectangle $R_{i,j}$. (Figure 7.8 shows the colored TC diagram for our example.) We call the colored rectangles **collision regions**. As we have seen, simultaneous execution of potentially colliding segments must be avoided. Thus, given a colored TC diagram, we can characterize all safe schedules as those that do not pass through any part of a collision region. Free areas of the TC diagram, by definition, represent non-colliding configurations of the two robots. Schedules which stay within these areas, then,

represent safe motions for the robots. Any schedule which ventures into a collision region represents simultaneous execution of potentially colliding segments, which we forbid.

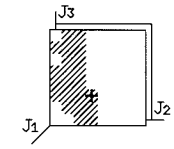


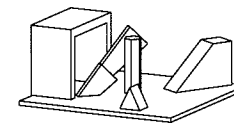
It is important to note that the boundaries between collision regions and neighboring non-collision rectangles are safe. (This is a consequence of Equation 7.3.1.) For example, consider the left boundary of a collision region, which is the same coordinate, a_i, as the right boundary of the neighboring rectangle. The setpoint represented by that coordinate, $\mathbf{q}_i = \mathbf{A}(a_i)$, belongs to both rectangles, so if there were a collision at that setpoint both rectangles would represent path segments which potentially collide. Both rectangles would have to be labelled as collision regions. Thus, since the left-hand rectangle is safe, the configuration $\mathbf{q}_i$ at the right boundary of that rectangle is necessarily safe. Formally, since $SV(A_{i-1}) \cap SV(B_j) = \emptyset$ and $\mathbf{q}_i \in A_{i-1}$, it must be the case that $SV(\{\mathbf{A}(a_i)\}) \cap SV(B_j) = \emptyset$.



Special considerations at the boundaries of the TC diagram The left and bottom edges of the TC diagram represent the beginnings of the paths of the robots. Before the paths begin to be executed, the robots are stationary in their initial configurations: $\mathbf{A}(a) = \mathbf{A}(a_0)$ for $a < a_0$, and $\mathbf{B}(b) = \mathbf{B}(b_0)$ for $b < b_0$. Any potential collisions with the initial positions of the robots must be represented in the TC diagram by collision regions. Since we can consider the initial configuration of the robot as existing "for all time" before the motion, these collision regions could be displayed as infinite rectangles extending beyond the boundaries of the TC diagram, acting as impenetrable walls. This argument applies equally well the the top and right edges of the TC diagram, where the robots' paths end. In practice, it is much simpler to add path segments at the beginning (and end) of each robot's path with no actual motion, duplicating the initial setpoint. In the TC diagram, this adds a row of rectangles just at the boundaries to represent the initial positions of the robots, and indicate the collision regions in these rectangles. These rectangles represent a path segment that includes no motion—the initial and final setpoint of the segment are the same point. It must be understood that these "edge" rectangles conceptually extend to infinity. We introduce them as an artifact to be able to represent collisions at the initial *setpoint* of the path, distinct from collisions during the initial *segment* of the path.

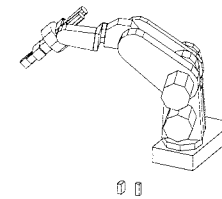
This interpretation of the edges of the TC diagram is important in the understanding of the boundaries of the collision regions, which are safe from collision, as mentioned above. Since a collision region at the edge of the TC diagram actually extends to infinity, the displayed edge in the diagrams is not safe (except for the origin and goal, both of which are assumed safe).





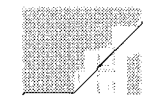
7.3.3 Deadlock and SW-closure

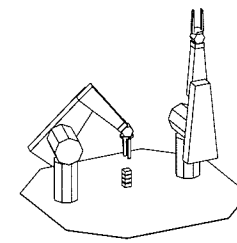
Our approach to multiple robot path scheduling was based on a technique for concurrency control in databases developed by Yannanakis, Papadimitriou and Kung [84], and illustrated in [69]. The key similarity to database scheduling is the concept of a **critical region** controlling access to a particular record in the database system, and access to a particular region of space in the robot control system. In the database system, when one party begins a transaction with a particular record, say adjusting a bank balance after a deposit, that record is locked, preventing any other party from accessing it until the first transaction has completed. This enforces a consistency in the bank's records. In the robot control system, whenever a robot begins to execute a portion of its path, the region of space through which that robot will pass is "locked," preventing other robots from entering it until the first robot has completed execution of that portion of its path.



A graphical interpretation of database locking schemes, presented in [69], inspired the multi-robot planner. The task completion diagram is used to schedule the trajectories of the robots to avoid both collision and deadlock and to guarantee a completed task.

It is fairly clear that safe schedules ensure that there are no collisions. It is not so clear, at least immediately, how to find safe schedules in a TC diagram. One simple algorithm could be described as follows:



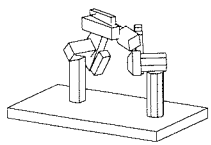


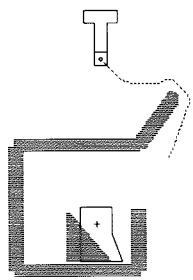
Step 1 Start at the origin.

Step 2 If you've reached the goal of the TC diagram, then stop. You have a complete, safe schedule.

Step 3 Proceed diagonally until you reach the boundary of a collision region or the edge of the TC diagram.

Step 4 If you're at the edge of a collision region, proceed along the edge (either horizontally or vertically) until you can continue on the diagonal. Return to step 2.





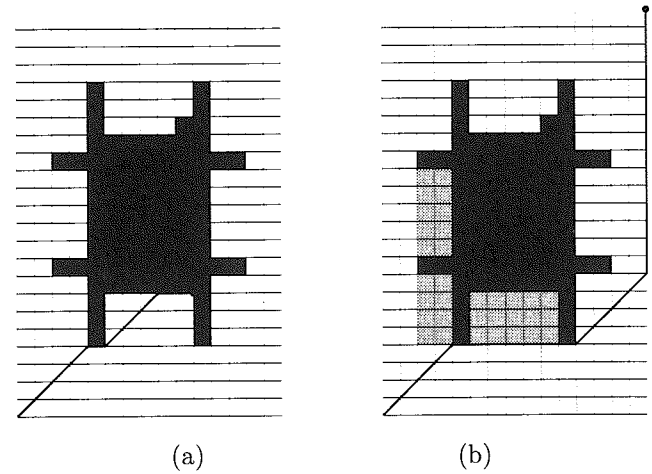


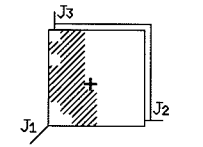
Figure 7.9
(a) A partial schedule that leads to a deadlock. (b) SW-closure of collision regions is shown lightly shaded.

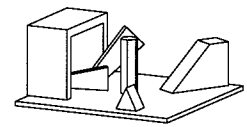
Step 5 If you've reached the top or right edge of the TC diagram, continue horizontally or vertically. If you reach a collision region while doing so, stop. The schedule has failed.

This algorithm, which we might call the "Greedy Scheduler" tries to move both robots together as much as possible. It prevents the schedule from entering any collision regions. However, as can be seen in Figure 7.9(a), the algorithm can run into a situation where it cannot proceed. Since the schedule is monotonic in both axes, it can only move up or to the right (but see Section 7.6.6). When the algorithm cannot move in either of these directions it has reached a **deadlock** or impasse. This corresponds to a situation in which neither robot can proceed along its path without the other arm getting out of its way (recall Figure 7.4(b)). Any continued motion would result in a potential collision.

We can modify the TC diagram such that the Greedy Scheduler always finds a deadlock-free, as well as collision-free, schedule. Notice that deadlocks can only occur at a concave corner facing the origin of the TC diagram. An origin-facing concave corner is a pair of setpoints (a_i, b_j) such that $R_{i,j}$, $R_{i-1,j}$, and $R_{i,j-1}$ are all collision regions. Only in such a corner can the Greedy Scheduler fail to move (in the TC diagram) either to the right or up. We can eliminate these concave corners by

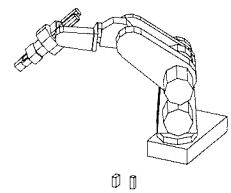
filling them with **pseudo-collision regions** (see Figure 7.9(b)). These regions act exactly like collision regions, in that we don't allow our scheduler to enter them, but they do not arise from collisions between the moving robots. The collision regions with the filled-in origin-pointing concavities are collectively known as the **SW-closure** [69] (for South-West closure) of the collision regions. We can now define a **safe, deadlock-free schedule** as any schedule which does not enter the SW-closure of the collision regions.



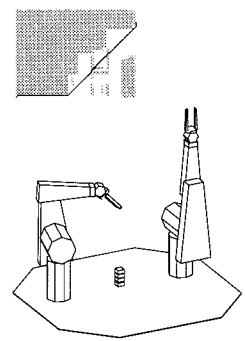


(Once again, the edge of the TC diagram requires special treatment. Since the edges of the TC diagram are not safe (see Section 7.3.2), any collision region touching the top or right-hand edge creates an origin-pointing concavity which must be filled for the SW-closure.)

Once the SW-closure of the TC diagram is taken, a safe, deadlock-free schedule exists if and only if both the origin and the goal are each not part of any collision region or SW-closure. Clearly, if there is a collision at the goal, the task cannot be completed; similarly for a collision at the origin. Also, if the origin is included in the SW-closure of some collision region, then there is an unavoidable deadlock, and again, a safe schedule is impossible. To show the converse, assume that the origin and goal are both clear. For there to be no schedule, there would have to be a connected collision region (including SW-closure) cutting across the entire TC diagram. But, it's clear that such a region must include the origin, since it touches either the right-hand or top edge of the TC diagram. Put another way, the safe areas including the goal and the origin must be connected.

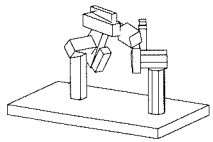


Let us assume, for the moment, that the origin is contained in a SW-closure, and thus there is unavoidable deadlock. By replanning part of the path of one robot using the swept volume of the other robot as an obstacle, assuming that we can find a new path to avoid this obstacle, we can guarantee that we can find a schedule to complete the task. This technique is described in greater detail in Section 7.6.1.



In some of the analysis of the TC diagram, it is convenient to also compute the **NE-closure**. This is similar to the SW-closure, but, as the name implies fills in concavities open to the northeast direction. These portions of the TC diagram are inaccessible to any schedule (because of the monotonic nature of a schedule), and it is simpler to just include them in the set of collision regions.

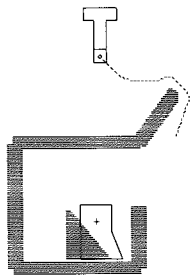
7.4 Generating the TC diagram



The preceeding sections described the motivation for the design of the multi-robot planner and TC diagrams. This section describes in more detail the construction of a TC diagram and its use. The two key steps in the multi-robot planner are:

1. Detect pairs of path segments which represent potential collisions between the robots.
2. Compute the SW-closure of the TC diagram.

The collision regions and the SW-closure guarantee finding a collision-free, deadlock-free schedule, if one exists. The rest of the multi-robot planner adds minor features and optimizations.



7.4.1 Finding the collision regions

Finding collision regions in the TC diagram is straightforward. For each pair of path segments, A_i and B_j, one simply compares the volumes of space swept out by the robots, the **swept volumes**. If those volumes intersect, $SV(A_i) \cap SV(B_j) \neq \emptyset$, then there is a potential collision and we have a collision region. Figure 7.10 shows an example of this computation. This test is performed for every pair of segments from each robot, and all the collision regions in the TC diagram are found. The multi-robot planner explicitly generates a representation of the swept volumes of the path segments, then tests these volumes for intersection. Note that we do not need to actually compute an intersection of the swept volumes; we only need to test whether they intersect. (In Section 7.4.6 we mention other work which can detect collisions on path segments without these explicit intersection tests.)

We have been saying that a collision region implies a *potential* collision between the two robots. The intersection of a pair of swept volumes does not imply that a collision *will* occur if the segments are executed simultaneously. It only implies that a collision *may* occur. The intersection of the swept volumes does not generally involve the whole of either swept volume; the collision would only occur if the controllers for the two robots brought them both to the area of physical intersection at the same time. Other timings could, in fact, avoid the intersection and thus avoid the collision. However, since we assume that the controllers are

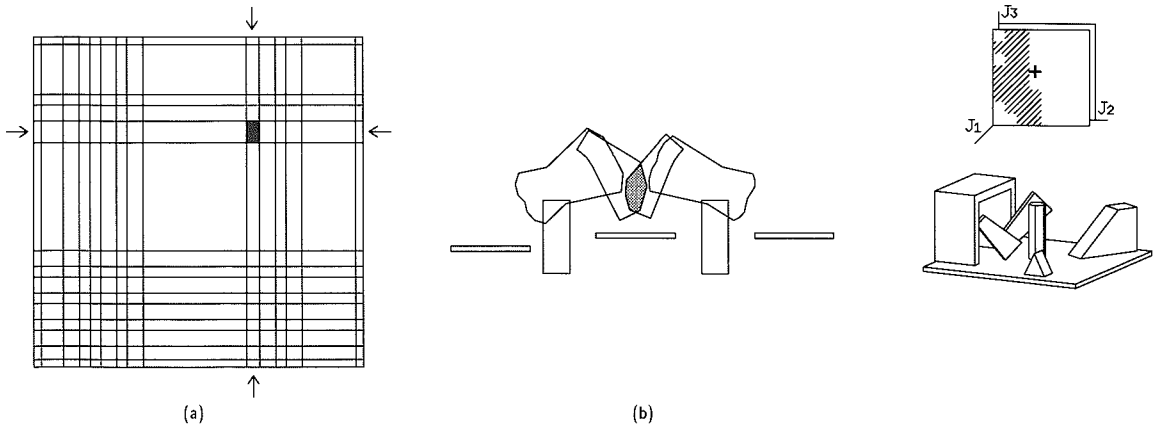
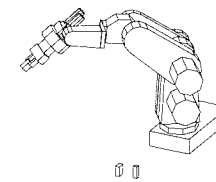
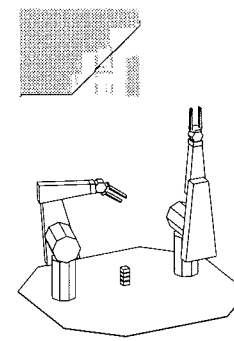


Figure 7.10
(a) Our TC diagram, indicating the row and column corresponding to the path segments being tested for collision. (b) The swept volumes of the path segments being tested. Since they collide the grid rectangle shown in (a) will be colored.



not perfectly controlled, we cannot hope that they will serendipitously choose a timing which avoids collision. (We will see in Section 7.4.6 that the swept volumes we compare are actually approximations; the true swept volumes may not even intersect.)

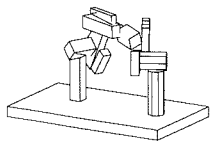
7.4.2 Finding the SW-closure



It is quite simple to generate the SW-closure. An algorithm in [69] will compute it taking time proportional to the number of collision regions in the TC diagram. A slightly less efficient but much simpler algorithm is described here.

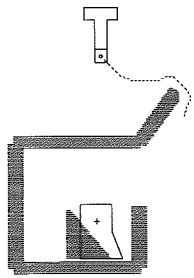
We start at the goal of the TC diagram, and proceed down the each column. At each rectangle, $R_{i,j}$, if the rectangle above, $R_{i,j+1}$, to the right, $R_{i+1,j}$, and to the upper right, $R_{i+1,j+1}$, are all collision regions, then make the current rectangle a pseudo-collision region. (It is a *pseudo*-collision region because it does not represent a potential collision, but the rectangle must be avoided anyway to prevent deadlock.) The procedure is then repeated for the next rectangle down. (Note that pseudo-collision regions also spawn further pseudo-collision regions if they create origin-pointing concavities.) When each column is

completed in this way, we proceed to the next column to the left. (As mentioned before, since the edges of the TC diagram are considered not safe, a collision region on the top ($j = n$) or right ($i = m$) edge will force the addition of a pseudo-collision region.)



7.4.3 Optimizations

It is not actually necessary to compute the entire TC diagram. There are several ways to avoid some of the computation; this section discusses two of those ways.



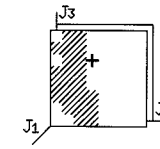
Schedules which are predominantly diagonal naturally include more motion with both arms moving simultaneously. Schedules which are predominantly horizontal and vertical include more serial operations with one robot moving, then the other. Clearly, diagonal schedules are preferred. If we assume that there is a safe schedule near the main diagonal of a TC diagram, then we can save time by only testing for collision regions near the diagonal. We treat any part of the TC diagram which has not been tested as a pseudo-collision region. If no safe schedule can be found in this limited search, we continue testing areas farther away from the diagonal until a schedule can be found or until the whole TC diagram has been completed.

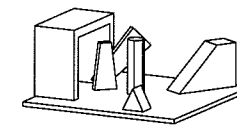
Limiting the testing and construction of the TC diagram is a very potent method to speed up the operation of the multi-robot planner. The test for potential collision is, in general, expensive, and it is the key operation in the inner loop of generating the TC diagram. Any heuristic which avoids intersection testing will save much work. Limiting the search for a schedule to be as close to the diagonal is one such heuristic. Another is incrementally computing the NE- or SW-closure of the TC diagram.

Let us assume for the moment that we test for collision regions column by column from left to right (increasing i), and in each column, we test rectangles from bottom to top (increasing j). Consider now a situation in which we are about to test a rectangle which would be part of the NE-closure of the partial TC diagram we have already computed; that is, the rectangles below, to the left, and diagonally below-left are all collision regions. There is no schedule which can enter the current rectangle, so there is no need to actually test the rectangle. We can simply tag it as a pseudo-collision region just as if it were in the NE-closure. (If we generate the TC diagram in the opposite direction, from the goal to the

start, we can perform the same optimization—incrementally computing the SW-closure.)

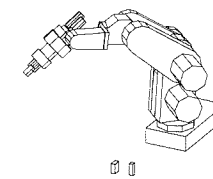
Another useful heuristic to limit the portion of the TC diagram to be tested is the NE-closure of another type of pseudo-collision region—regions generated due to explicit synchronization constraints between the robots.





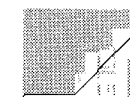
7.4.4 Synchronization between robots

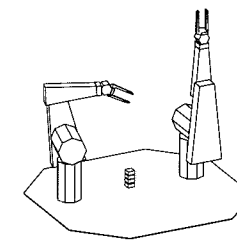
When multiple robots work together in a common workspace, they frequently are required to interact in such a way that their operations must be synchronized. For example, one robot may grasp an object that the other robot has put down in a temporary location—the first robot cannot perform the grasp before the second one has placed the object down. Such synchronization requirements constrain the possible schedules that we can execute to perform the required task. Although a TC diagram with collision regions and NESW-closure will guarantee that all safe schedules are collision- and deadlock-free, it does not guarantee that all safe schedules actually perform the desired task.



We can add pseudo-collision regions to enforce *a priori* synchronization constraints which are imposed by the task itself. These new regions will have the effect of blocking out entire sections of the TC diagram, since, in the example just mentioned, the first robot cannot execute any of its path subsequent to the grasp until the second robot has executed its path up to the point of ungrasping the part at its temporary location.

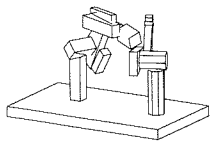
Let us say that robot A ungrasps a part during segment A_i, and robot B then grasps that part during segment B_j. Then robot B cannot execute segment B_j (or any subsequent segment) until robot A has completed segment A_i. We capture this constraint by introducing pseudo-collision regions at $R_{i',j}$, for each i' such that $0 \leq i' \leq i$. This prevents any safe schedule from allowing the waiting robot to proceed beyond the point where it must wait for the event (see Figure 7.11).

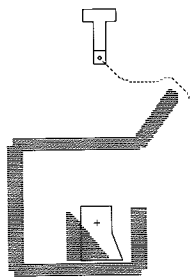




7.4.5 Three-dimensional robots

Although we have been illustrating this discussion with a situation involving planar robots, there is nothing in the TC diagram which depends on the physical structure or degrees-of-freedom of the robots. For two robots, the TC diagram will always be two-dimensional, and





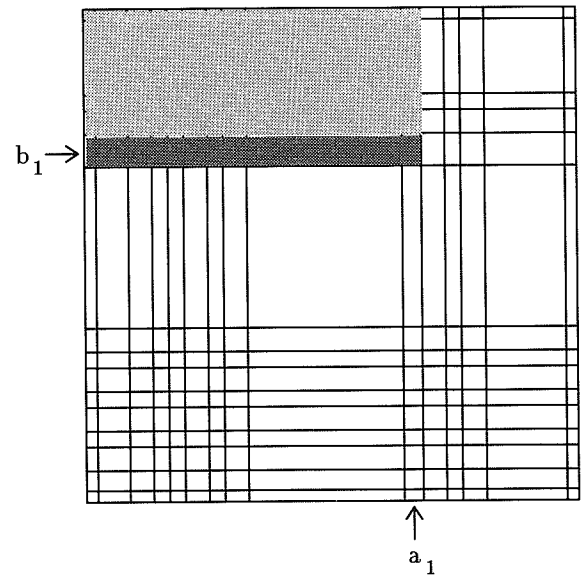
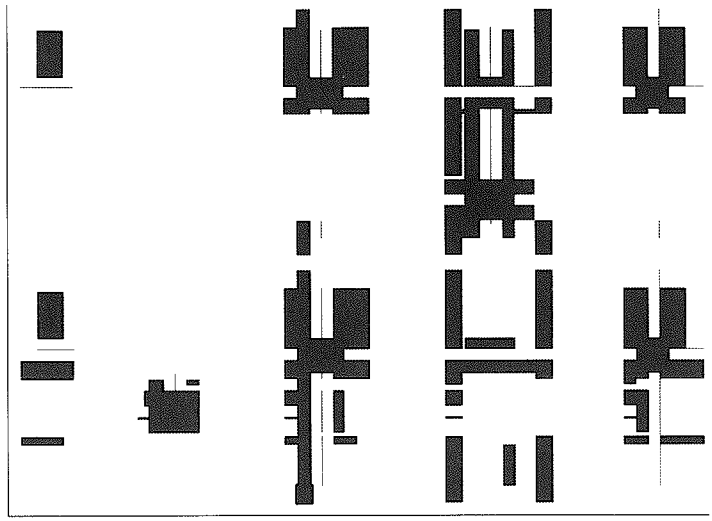
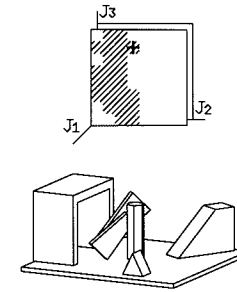


Figure 7.11
Pseudo-collision region representing an *a priori* synchronization constraint between two robots. Since robot B cannot proceed past b_1 until robot A has completed its motion up to a_1, the pseudo-collision region blocks all schedules from allowing that possibility. The NE-closure completes the representation of the constraint.

will be constructed in the manner described above. The only portion of the multi-robot planner which depends on the physical robots is the comparison of the path segments for detecting collisions, specifically, the computation of the swept volumes and the test for intersection of those volumes. The swept volume code is completely separable from the rest of the multi-robot planner (and as mentioned in Section 7.4.6, could be replaced with a functionally equivalent module that does not explicitly compute the volumes).

In Right-Margin Movie 3, we demonstrate an entire task planned and executed using the multi-robot planner. (See the description of the Margin Movies in "On the Margins," page xvi.) The task was to build a short stack of parts representing a common assembly. The parts all arrive (as if delivered on a conveyor) at a specific point in the workspace accessible only to robot A (on the right). They do not arrive in the order





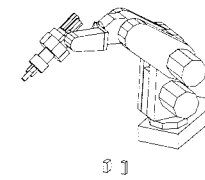
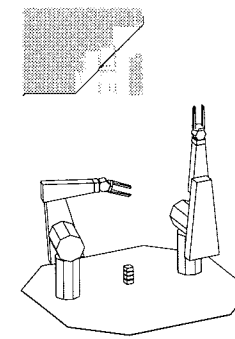
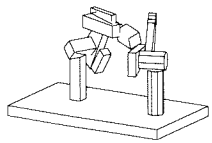


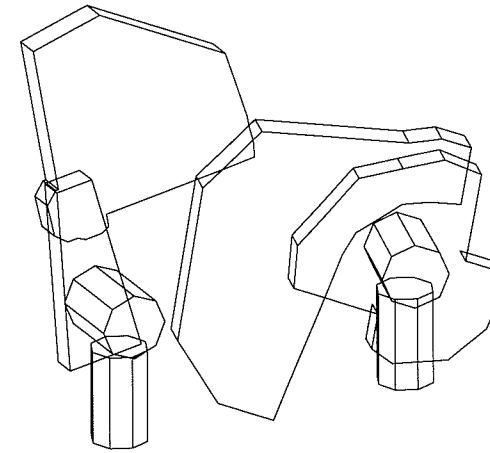
Figure 7.12
Collision regions in the TC diagram for a three-dimensional robot coordination problem. The potential collisions cluster around those portions of the robots' paths which lead them into the common area of the workspace. There are wide avenues where each robot is working in its "own" portion of the workspace. The swept volumes corresponding to the indicated collision region are shown in Figure 7.13.

necessary to be assembled, so some of them are placed in temporary holding locations. The two robots cooperate in assembling the parts. The paths for these robots were planned using the gross motion and grasp planners of HANDEY, and the two paths were coordinated using the multi-robot planner. The TC diagram for the task is also shown in the movie, along with the schedule used to coordinate the robots. As the task proceeds, the line representing the schedule is drawn.



Figures 7.12–7.16 depict the TC diagram corresponding to the task. Figure 7.12 shows just the actual collision regions representing true potential collisions. Note that there are several segments in robot A's path which cause many potential collisions with robot B. These particular motions of robot A are large sweeping motions through the common workspace, near robot B. The swept volumes for one such motion is





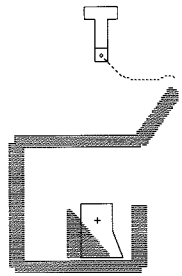


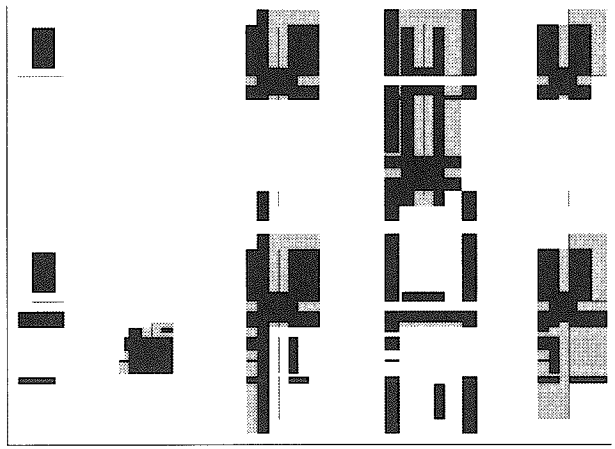
Figure 7.13
Swept volumes for one particular collision region in the TC diagram in Figure 7.12. The large motion of the right-hand robot into the common work area causes several potential collisions. A better path would reduce the number of potential collisions, and thus increase the potential parallelism in the task.

shown in Figure 7.13. In Section 7.6.5, we will discuss one method for alleviating some of the effects of these motions.

Figure 7.14 shows the SW- and NE-closure of the collision regions. A schedule in this TC diagram would be safe and deadlock free.

Figure 7.15 shows the synchronization constraints in the task. Two of the parts in this particular example are handled by both robots. Robot A transfers the parts from the "conveyor" to a temporary holding location, then robot B carries them to the final assembly once the parts which will support them have arrived. For both these parts, robot B cannot pick them up until robot A has placed them in their temporary locations. This accounts for two of the synchronization constraints. The other three occur as a result of the stacking assembly: no part can be added to the stack until the part below it has been correctly placed.

Finally, Figure 7.16 shows the complete TC diagram as computed by the multi-robot planner. One of the possible schedules is shown—the one computed by the Greedy Scheduler. It is this schedule shown in Right-Margin Movie 3. Notice in this diagram the SW-closure preventing robot B from proceeding very far before having to wait. If it were to



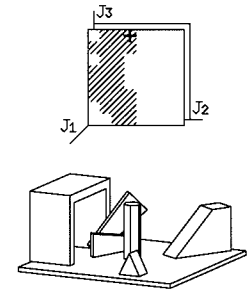
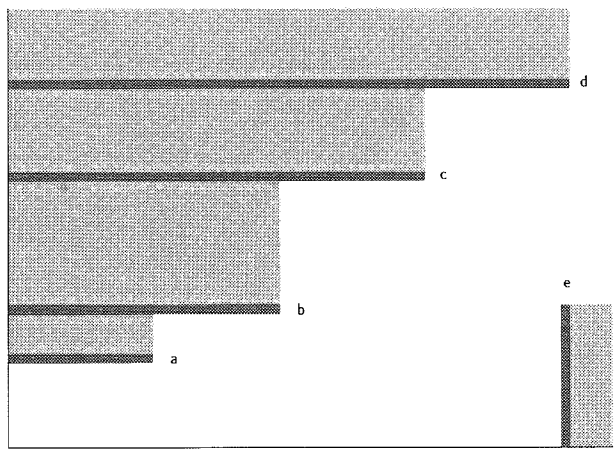


Figure 7.14
The same TC diagram as in Figure 7.12, with the SW- and NE-closure.



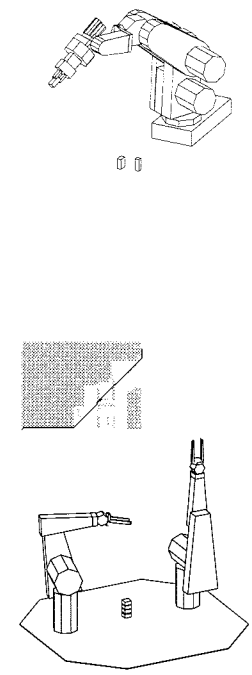
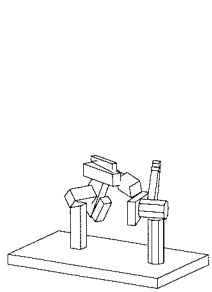
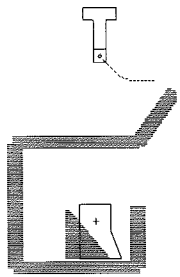


Figure 7.15
The synchronization constraints for the three-dimensional example. Two different types of synchronization are represented. Two parts are handled by both robots, requiring the second one to wait for the first one to ungrasp the part before grasping it (the synchronization constraints labelled a and c. Since the robots are stacking parts, each part must be placed on the stack before the one above it can be placed (the constraints labelled b, d, and e).





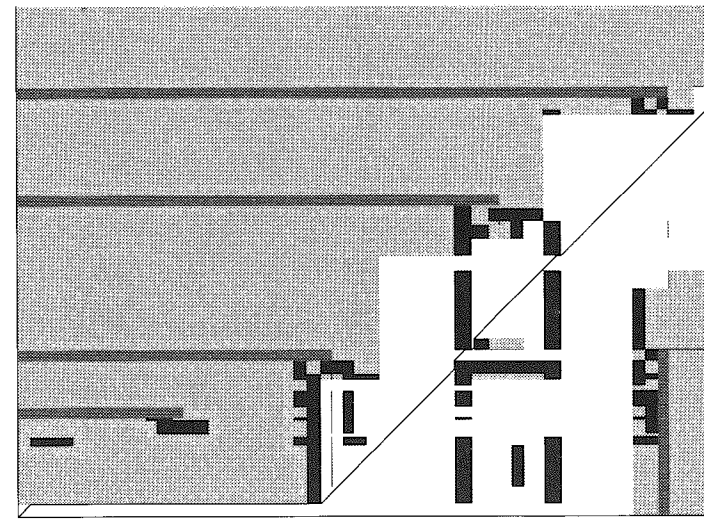


Figure 7.16
The final TC diagram for the three-dimensional example, including a possible schedule.

continue (avoiding collisions), it would reach a point where it was waiting for robot A to place the first part to arrive at its temporary holding location (the first synchronization constrant). Robot A, however could not proceed without colliding with robot B. The SW-closure prevents this deadlock and several others like it in this diagram.

7.4.6 Computing swept volumes

Our approach requires that we be able to detect potential collisions between path segments. That is, we need to identify which $R_{i,j}$ in the TC diagram need to be shaded. This is readily accomplished by computing the volume swept out by each manipulator while executing segment A_i and segment B_j and testing these volumes for intersection.

In HANDEY, we explicitly compute a representation for the swept volume. There are other collision-detection methods which do not require forming an explicit geometric model of the volume. Canny [9] describes a method of collision detection which uses a purely algebraic formulation

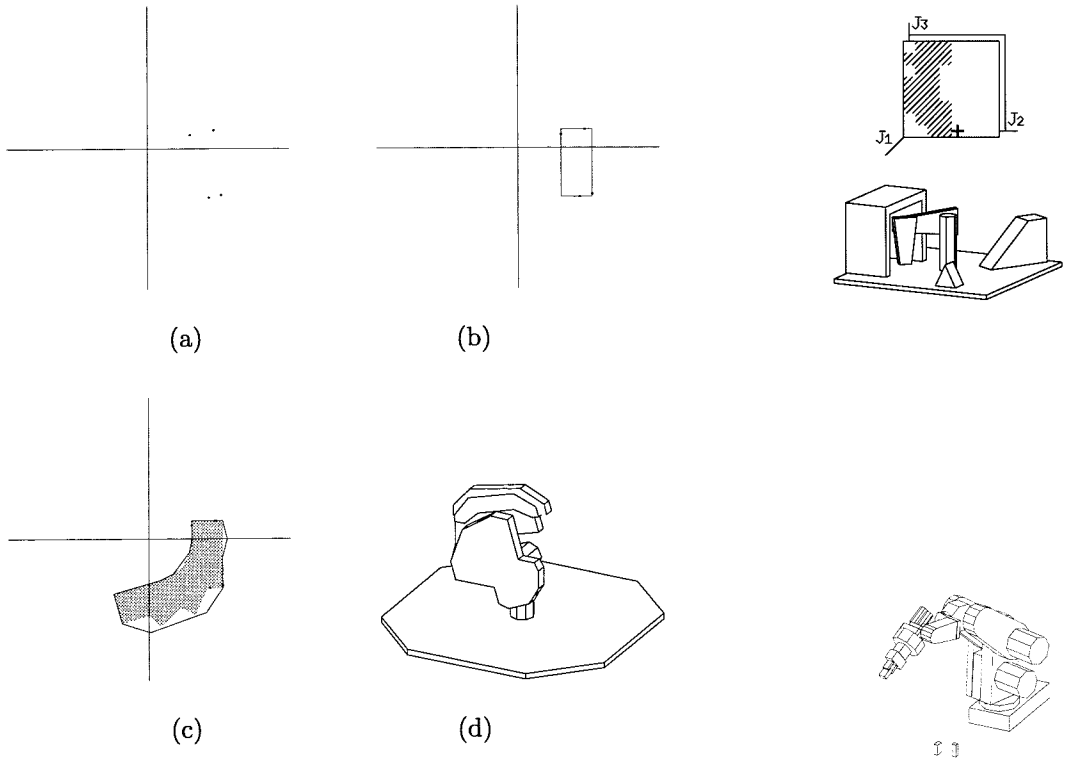
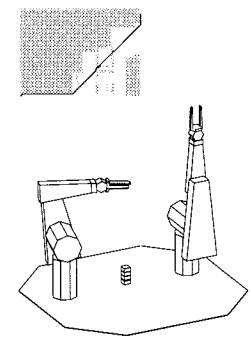
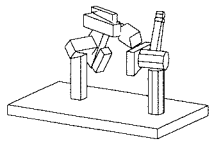


Figure 7.17
Steps in computing the swept volume of one link of a manipulator: (a) Link vertices, (b) bounding box, (c) sweep, (d) final result.

of the constraints between a moving polyhedron and obstacles. The solution of these algebraic constraints identify the potential collisions between the object and the obstacles. This method could be adapted to our current problem. The exact method of detecting the potential collisions is immaterial; this module can be replaced with another which is functionally equivalent.

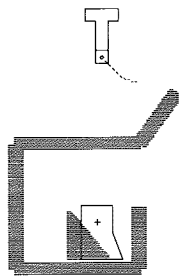
In general, it is hard to compute the exact volume swept out by the robot during a move, since it is non-convex and contains curved surfaces. In our implementation, we only compute an approximation of the volume swept out by an approximation to the links of the manipulator. In practice, this approximation works out very well.





Let us consider robot A and assume for the moment that only one joint is moving during segment A_i. For each link of the manipulator distal to the moving joint, we compute a rectangular bounding box for the link in the coordinate system of the moving joint. The problem now reduces to finding the swept area of a two-dimensional rectangle in the xy-plane of the moving joint, and sweeping the resulting polygon between the z-coordinates of the bounding box.

Figure 7.17 illustrates this procedure. Link 3 of a Unimation Puma robot is shown being swept through 74 degrees of rotation of joint 2. The vertices of the polyhedral model of the link are projected onto the xy-plane of the coordinate system for joint 2. The bounding box for these projected points is computed, then a polygonal approximation to the swept area of that rectangle is found. The final diagram shows the entire swept volume for links 2–6.

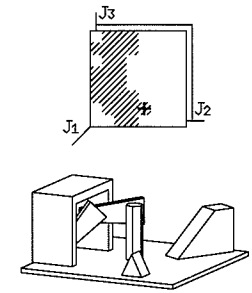


It is clearly possible to approximate the true swept volume of the bounding box of the link arbitrarily closely. It is also possible to improve the algorithm so as to approximate the actual link arbitrarily closely, and thus compute a polyhedral approximation to the swept volume with better and better accuracy. However, detecting collisions between the swept volumes takes time proportional to the product of the number of edges and faces of the polyhedra, so unless there is an overriding reason to use very accurate swept volumes, it is beneficial to use simpler polyhedra (rougher approximations) to save time in collision detection.

The procedure above demonstrates the construction of the swept volume for one moving joint, and only for a revolute joint. The swept volume of a prismatic joint is trivial to compute exactly. It is not so clear how to extend the procedure to multiple moving joints. The approach we have taken is to use the volumes swept out by distal links due to the motion of the distal joints as "virtual links" when computing the swept volumes of the proximal joints. For example, if both joints 2 and 3 are moving, we first compute the swept volumes of links 3-6 due to the motion of joint 3. We then substitute these polyhedra for the actual links 3-6 and compute the swept volume due to joint 2.

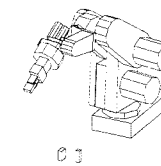
This procedure for handling multiple moving joints is extremely conservative, as it computes a much larger volume than the robot is typically going to sweep out. But, this approximation is consistent with the

mimimal assumptions we have made on the shape of the paths, namely, that all joints stay within the limits specified by the endpoints of the path segment. Strict coordination between the joints is not necessary.



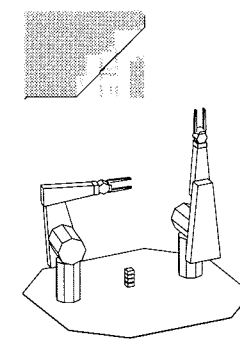
Detection of a potential collision is achieved by simply testing whether the polyhedral approximation to the swept volumes of A_i and B_j intersect. The actual volume of the intersection does not need to be computed. Since an intersection test between two arbitrary polyhedra can be expensive, we have implemented several quicker tests to determine if the full swept volume approximation needs to be used.

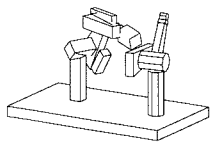
Before the actual swept volume is computed for A_i, we compute a bounding box approximation to the swept volume. This can be done much more quickly than computing the full approximation. What is actually calculated is the bounding box in world coordinates of the bounding box in joint n coordinates for the swept volume due to motion of joint n. Only if there is some B_j for which the bounding boxes intersect is the full swept volume for A_i computed, and the intersection test of the full swept volume is only performed between those A_i and B_j for which the bounding boxes intersect.



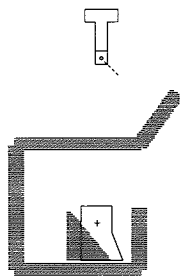
7.5 More on schedules

There are two general approaches to constructing a schedule, given a TC diagram. One is a local method, such as the Greedy Scheduler shown earlier. That particular scheduler assumes that there is a central controller that initiates the motions for both manipulators. One can also build a decentralized version of the greedy scheduler for the common case of independently controlled robots. In that case each collision (or pseudo-collision) region becomes a "lock," that is, a variable that can be indivisibly tested and set so that we can guarantee that only one process "owns" the variable. Before executing a path segment, say A_i, robot A's controller must grab the locks of the collision regions formed with each of the path segments for robot B, B_j, for all j. Similarly, before executing path segment B_j, B's controller must grab the locks of the collision regions for all i. In this scenario, the locks corresponding to collisions and to the SW-closure must all be obtained. (Of course, one may actually aggregate adjacent locks in a column or row into a single lock if desired.)





An alternative approach to scheduling, which we can call global, involves searching the TC diagram for a schedule that is "optimal" by some measure, for example, the total execution time. This global search can also guarantee finding a legal schedule if one exists without the need to assume that the purely sequential schedules are safe. A schedule, such as may be found by this search, corresponds to a fixed sequence of activations for each of the path segments. A schedule can be characterized by the sequence of its crossings of the horizontal and vertical lines that bound the path segments in the diagram. Crossing each line adds to the schedule a command to wait for the completion of one segment and then to initiate the next segment. Such a schedule can be implemented in a centralized controller simply by marching down a list and issuing the appropriate START and WAIT commands. A decentralized implementation is also straightforward.



With either scheduling approach, global or local, an important point must be made. The schedules we have been discussing have all specified a precise relationship between the execution of the paths of the two robots. In the notation of Section 7.3.1, the trajectories of the robots are related by the formulæ $a = f(s)$ and $b = g(s)$. Requiring strict adherence to any of these schedules, however, violates two of the goals we have set for the multi-robot planner: non-dependence on accurate trajectory control of the individual manipulators and lack of precise time coordination between the manipulators. In essence, these goals say that we cannot specify the details of the functions f or g.

What we can specify is a series of **synchronization points**. (These are *not* the same as the synchronization constraints which generate pseudo-collision regions in Section 7.4.4.) These points in the TC diagram specify conditions which require one of the robots to wait until the other robot has completed some portion of its path. This is very similar to using the locks as described at the beginning of this section.

Another way of looking at synchronization points requires looking more closely at how schedules and actual execution on the robots interact. When we give a setpoint from a robot's path to the robot controller, we cannot predict the exact time course of the execution. The actual execution will indeed be represented by some schedule in the TC diagram, but not one we can specify in advance. That schedule, however, will be entirely contained in the rectangle formed with the current position of the robots in the lower left corner and the new setpoints in the upper

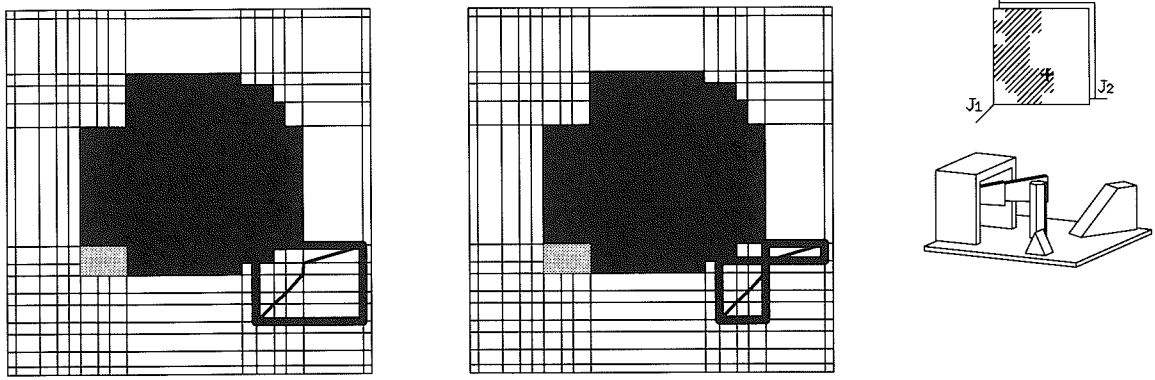
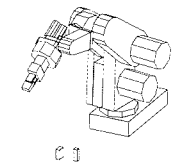
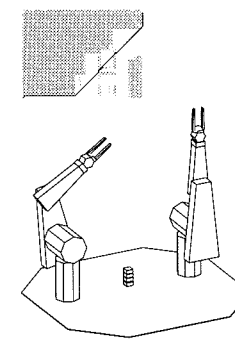


Figure 7.18
Schedules actually executed by the robot controllers cannot be predicted in advance. Such schedules will be contained in the rectangle formed by the initial and final setpoints of the segments being executed. By ensuring there are no collision regions within such rectangles, we can be assured of safe schedules.

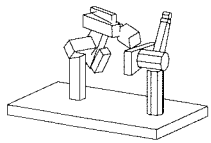


right corner.[1] We can supply setpoints to the robot controllers as long as that rectangle does not contain any collision regions. That constraint is sufficient to ensure that whatever actual schedule is produced by the execution of the paths is safe, since no such schedule will enter a collision region (see Figure 7.18).

To avoid collision regions with unpredictable schedules, we can proceed in two ways. (The two ways are essentially equivalent, but they both provide interesting ways of understanding synchronization points.) The first way is to use the locking scheme mentioned earlier. We have already seen how that method prevents entry into collision regions. The second way is to find the largest rectangle with the current position at the lower left corner which is clear of collision regions. The upper right corner of this large rectangle becomes a synchronization point.



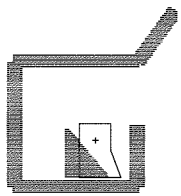
[1] We assume that the controller is capable of at least this: that it will not cause any joint to overshoot its final joint angle in the goal setpoint, and that it will not cause any joint to "back up" beyond the initial setpoint. This is a fair assumption, though it should be noted that some control strategies may fail this assumption.



Command the robot controllers to execute the paths up to the top and right edges of this rectangle. Once the controllers have completed that much of the paths, repeat the process from the new current point.

Variations on either of these two methods are possible. The locking scheme is a local method, while the rectangle method is global. Various combinations of the two result in hybrid methods.

The synchronization point concept can be generalized to allow increased parallelism when one robot executes its path faster than the other. Note that the TC diagram generally leaves a choice whether to go "over" or "under" a given connected set of collision regions. Rather than finding the largest rectangle which is clear of collision regions to generate the synchronization point, we find the largest rectangle which still leaves us a choice of which direction to go to avoid a connected set of collision regions. Once we reach the synchronization point we can choose the direction to proceed which will give the best possibility of finishing the task quickly.



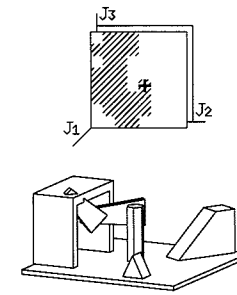
7.6 Other issues

This section discusses details that are not currently implemented in the multi-robot planner, but which require mention to fully develop the concepts introduced by this chapter. In particular, there are further manipulations of the TC diagram which allow the parallelism in many tasks to be improved, based on the results of the initial TC diagram construction. We must address the issue of parts that are manipulated by the robots, since they move around in the environment. The TC diagram can be improved by adding constraints on the paths provided as input to the multi-robot planner. We also discuss the implications of removing our requirement that robots do not retrace any portion of their path, and finally we describe how to extend our algorithm to enable coordination of more than two robots.

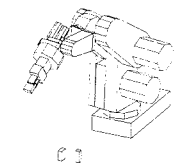
7.6.1 Increasing parallelism

In the preceding discussion we have largely ignored the issue of the time required to execute a schedule. In practice, time is crucial. In what follows, we will assume that the axis parameters of the TC diagram are designed to correspond to expected execution time. Each

path segment will have an expected time and this will determine its dimension in the diagram. Given this TC diagram, we can search for a schedule with the best expected execution time. The best possible schedules tend to have a great deal of parallelism, that is, they are nearly diagonal lines in the TC diagram. But, a particular TC diagram may have many collision regions near its diagonal, forcing the "best" schedule into the sequential execution of large segments of the path. The fault is not in choosing the schedule but in the original choice of paths. If the paths were chosen completely independently, there is no guarantee that much parallelism is possible. It is possible, however, to take two paths and to increase their parallelism by modifying some segments of the paths.

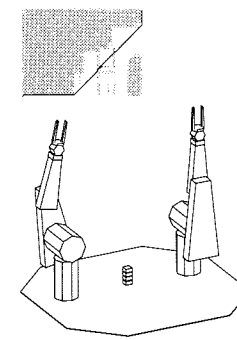


One modification which is can achieve good results is to split segments by introducing intermediate setpoints. If the potential collision results from an intersection in only a small portion of the swept volume, then this will create a new segment (or segments) with no collisions as well one (or a few) which still have a potential collision. This would be especially true if the path planner generated paths with large sweeping motions (see Figure 7.13, page 190).



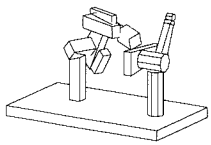
Another modification is to replan a portion of the paths of one or the other robot. The resulting TC diagram will allow more parallelism but the paths will generally be longer and may, therefore, increase the total execution time. To increase the potential parallelism in a TC diagram, we pick a collision region, or a larger region formed from the union of several collision regions, such that:

1. the region is shaded because of a collision and not because of the SW-closure operation,
2. it is near the main diagonal of the TC diagram, and
3. the region is large enough that it causes a significant increase in the total time of the best schedule to go around it.



Having chosen one or more of these regions, new paths can be planned to connect the initial and final points of robot A's segments, but using as obstacles the volume swept out by robot B as it moves through its segments.

A safe path may not exist but, if it does, this means that the region in the new diagram will no longer need to be shaded. Of course, the new path may introduce collisions with some segments that may not have



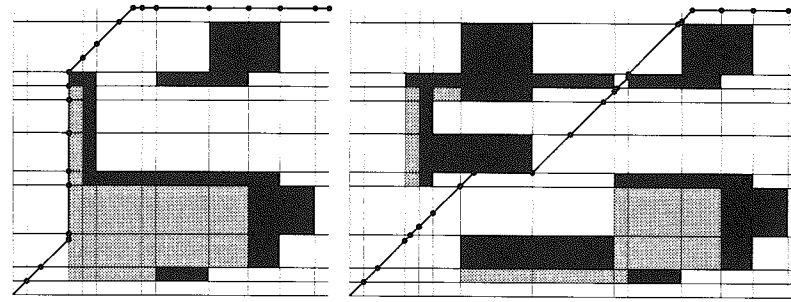
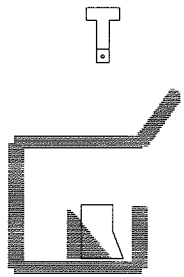


Figure 7.19
An example illustrating the increase of parallelism in a schedule, obtained by clearing a collision region. Segment A_4 in the left diagram has been replaced by a new three-segment path in the right diagram. Note that the new path for A has longer expected time, but the increased parallelism results in a schedule that is somewhat faster overall.

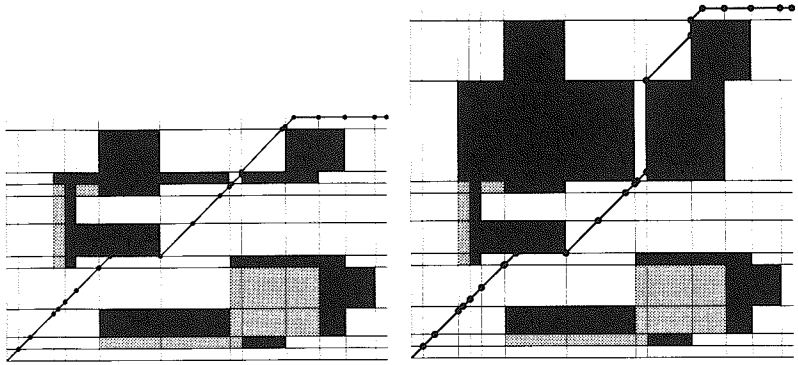


previously collided. Fortunately, such collisions will be off the diagonal and therefore will not be likely to affect the desired schedule. Also, the new path will generally be longer than the original path since there are new obstacles to be avoided. On the other hand, the impact of clearing one collision may be greater than just clearing one small region due to the impact of the SW-closure.

The process of increasing parallelism by replanning paths is illustrated by the simple example in Figure 7.19. Note that by focusing on the collisions near the diagonal, we are engaging in a crude form of space-time planning. We focus on just the combination of segments that we want to be executed at the same time (to achieve the desired parallelism). This is very different from, and substantially better than, the trivial strategy of using the volume swept out by one robot over its complete path as an obstacle while planning the path for the other robot.

7.6.2 Dealing with variable segment times

Earlier, we indicated that in many applications, the execution times for path segments cannot be predicted reliably, especially in situations involving sensing or variable-time processes. What is the impact of the change in length of one of the path segments? The crucial impact is that it may change the choice of the best schedule, for optimal planning. Geometrically, the stretching of a segment may move some new collision region into, or out of, the path of the best schedule (Figure 7.20).



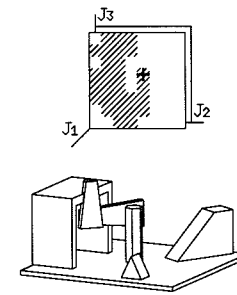
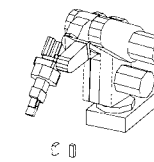


Figure 7.20
The effect of delay in the TC diagram is to move collision regions on or off the diagonal. This figure shows the effect on the schedule of increasing the time for segment B_8 between the left and the right figures.

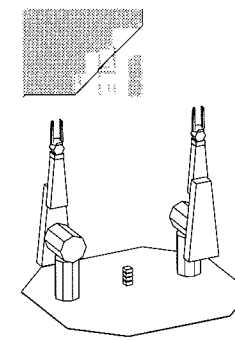
One simple strategy when faced with a significant delay is simply to redo the coordination of the remainder of the schedule in the modified diagram. It may be possible, however, to characterize the possible changes to the schedule brought about by different changes in the execution times of the various segments and to construct a decision tree that can be used on-line. We will be investigating this option in the future.



7.6.3 Changing the task

When there are substantial delays in the execution of one or more segments, it may be desirable to change the allocation of tasks from one manipulator to the other. In the preceding discussion, we have assumed that the task assignments are fixed, but this need not be the case.

Consider a task where four parts are to be taken from an input pallet, processed and taken to an output pallet. There are two processing stations, each with its own robot. The initial assignment has each robot doing two of the objects. What happens if robot A becomes delayed waiting at the first processing step, perhaps waiting for human intervention? Usually, we would like robot B to take over the processing of the other three parts.



Assume that the robots start each cycle in a standard position, so the last motion of a cycle is to return to that position; this assumption can be relaxed later. Then, we construct the TC diagram assuming that each robot will carry out the complete task, that is, process all parts.

This expanded diagram contains all the combinations of assignments of task steps to the different robots. Furthermore, in this TC diagram, schedules can jump from the end of one part cycle to the beginning of another one that may not be adjacent to it in the diagram. This jumping around is made possible by our assumption that the endpoints of each cycle are the same. If they were not, we would have to plan the transfer motions between each pair of cycles separately, but this does not present a fundamental problem.

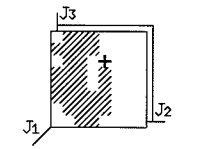
The method outlined above has the drawback of requiring separate planning of each part cycle, including its interactions with all other possible cycles. In practice, cyclical tasks tend to be mostly the same motions except for a few path segments, such as when picking up a new part from its own pallet location. We can construct a "generic" cycle path that represents the union of the path segments for all the cycles. This path union can be used to compute the swept volume of the robots over all instances of the cycle for different parts. We can then do the planning for the possible interactions of the two robots as if they were each executing only the generic path. For tasks in which the actions performed in each "cycle" are very different, one must consider each cycle separately using the expanded diagram suggested above.

7.6.4 Carried (moving) objects

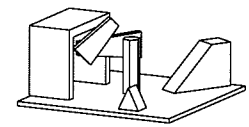
As we have stated before, the multi-robot planner considers only moving robots and fixed obstacles (and these only through the path planner). However, when multiple robots are cooperating to perform some task, they will be moving other parts around the workspace. These moving parts, though under the control of the robots, nevertheless create some problems for the path planner and for the multi-robot planner.

Consider a simple case where robot A carries part P from point X to point Y. For the robots to safely complete their task, they must both avoid colliding with the part while the part is at point X, while it is being carried by robot A, and while the part is at point Y. While it is being carried, we can simply treat it as part of robot A while testing for potential collisions, and the multi-robot planner will ensure a safe schedule with respect to robot B. We must, however, avoid collision with the part while it is sitting idle (and the robots are going about other business). We cannot simply include it as an obstacle for the main path planner module: do we place the part at point X or at point Y? Since

the robot's paths are planned independently, there is no way to predict where the part is going to be during any particular path segment for *either* robot. (Both robots may manipulate the part during the task.)

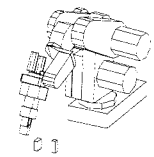


The multi-robot planner does not implement a solution for this problem. One potential solution, as far as the multi-robot planner is concerned, is to consider all the manipulated parts as "links" of another robot, to be avoided just as all the "normal" robots are. The TC diagram which represents the scheduling task will be constructed using the geometrical descriptions of the parts and their locations as they are moved about. Appropriate synchronization constraints will be necessary to coordinate the manipulation of the parts by the regular robots.



7.6.5 Path planning considerations

Throughout this chapter, we have maintained that the paths for the robots are planned independently, without taking into account the location or motion of any of the other robots in the workspace. If we're willing to forego some of this independence, we can take advantage of our global knowledge of the task to greatly improve the parallelism available in the robot's motions to speed the ultimate safe schedule's execution. In particular, if we prevent or inhibit one robot's path from entering space where the other robot is likely to be, the potential for collision can be greatly reduced, and the TC diagram will have few collision regions. This allows the multi-robot planner much greater freedom to identify safe, parallel schedules.



Stationary links as obstacles One portion of the workspace which we want to prohibit a robot from entering is that portion occupied by stationary links of the other robot. If we sweep a link of robot A through the base of robot B, then robot B will never be able to get out of the way, and the task cannot be completed. (This would manifest in the TC diagram by a collision region stretching from top to bottom.) We prevent this situation in HANDEY by including the base of robot B as an obstacle while planning the path for robot A (and vice versa).

Near the base of a robot is a region of space which the robot is almost always occupying, even if it will sometimes be moved away. By including a phantom obstacle surrounding this region during the path planning for the other robot, we can once again reduce the likelihood of potential collisions, and keep the TC diagram emptier.

Since most robots can reach a given point in Cartesian space in more than one configuration of joint angles, we can further reduce the potential collisions by requesting that the path planner choose configurations which are less likely to collide. Unlike the phantom obstacles just mentioned, which depend only on the design of the robot being avoided, choosing good configurations requires knowledge of the relative positions of the robots and their likely motions. This further requires some knowledge of the tasks the robots will be asked to perform. In practice, if we simply try to choose configurations which keep each robot's links furthest from the other robot, we succeed in minimizing the number of potential collisions. Note that unlike simply adding obstacles for the path planner to avoid, we would now be requiring more of the path planner. This would reduce the interoperability of the multi-robot planner with path planning modules which one might wish to try in HANDEY. Fortunately, the added interface requirements are not necessary for the multi-robot planner to work—they only improve its performance.

Task considerations While we're creating a wish list for the path planner, let us consider one more feature. The portion of the workspace which is most likely to create potential collisions is the common assembly area, which both robots will need to enter during their operation. It is quite clear that there will be many potential collisions here, and most of them cannot be avoided by clever path planning. However, if we can prevail upon the path planner to avoid the common assembly area *except when it must be entered,* then we limit the potential collisions to the barest minimum. In other words, we try to keep the robots from "just passing through" the assembly area on their way somewhere else. This could be achieved in a path planner by "gray" obstacles—objects that are avoided unless a path cannot be found without entering them.

7.6.6 Backing up on a path

In the preceeding sections of this chapter, we have assumed that execution of the robots paths were strictly one-way. In our coordinate notation, we have required that $f'(s) \geq 0$ and $g'(s) \geq 0$. By using this assumption, we have used the SW- and NE-closure to simplify analysis of the TC diagram, and we have been able to provide simple definitions

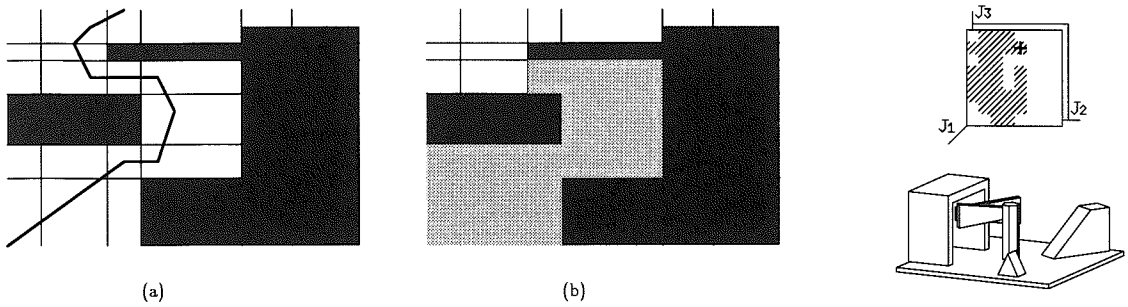
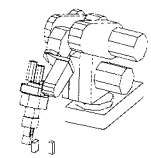


Figure 7.21
(a) A fragment of a TC diagram where backing up on robot A's path would be helpful. (b) The SW-closure of the TC diagram in (a).

of safe and deadlock-free schedules. This assumption is warranted by the one major goal of the multi-robot planner—to perform tasks faster by increasing the parallelism available by using multiple robots to perform a task. Forcing the robots to retrace a portion of their path to achieve the task is counterproductive to this goal. Nevertheless, we will now examine what it would mean to our analysis if we were to relax this assumption—if we were to allow the robots to go backwards on their paths.



It should be clear that relaxing this assumption will complicate the analysis of the TC diagram to some degree. To justify the extra work, we should try to identify situations in which backing up might be useful. Figure 7.21 shows a fragment of a TC diagram in which backing up allows us to find a schedule which would not otherwise be possible. In fact, if we were to generate the SW-closure of the diagram (Figure 7.21(b)), the rectangles traversed by the schedule would be pseudo-collision regions.

It should be clear that in any SW-closed TC diagram it would never be necessary to back up, though it may be the case that no safe schedule exists. Without the SW-closure, there may exist safe schedules that didn't exist before, but, because of deadlock, we cannot use any local method to find the schedules. Global methods would be able to avoid deadlock, and, in fact, the problem of finding schedules in such a TC diagram becomes one of finding a path for two-dimensional mobile robot among obstacles.

Side effects prevent back-up Now that we have motivation for allowing a robot to back up in its path, let us examine the feasibility of the robot actually doing so. The paths of the robots have been planned taking into account the obstacles of the environment. As long as the environment has not changed during the forward execution of the path, the backward execution should be completely safe. If some side effects have occured (such as a grasping or ungrasping operation by the robot), then the backward execution would not be the same as that which was planned, and would be susceptible to collisions. Thus, any backing up planned by the multi-robot planner must be restricted to portions of the paths which contain no side effects.

(Of course, if we undo whatever side effects are performed in forward execution in the backward execution, the time-reversal should be sufficiently equivalent to forward time that the planned path will suffice.)

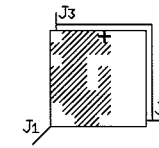
Some optimizations no longer valid When we allow schedules to be non-monotonic, we may no longer generate the SW- and NE-closures of the TC diagram. Of course, this means that we may not compute incremental closures of the TC diagram, and that collision tests may not be entirely avoided in this way. Other optimizations, such as diagonal searching, may still be implemented.

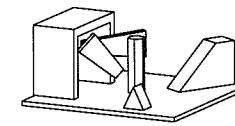
It must be emphasized here that the NE-closure of synchronization constraints must still be computed and obeyed. It does no good to wait for the other robot to reach a particular point in its path before proceeding if we then allow it to back up to before the point we required it to reach.

Fall-back analysis Allowing the robots to back up makes analysis of TC diagrams more difficult and it requires suppression of some useful optimizations. These disadvantages may be deferred in normal operation by generating the normal TC diagram as discussed in preceeding sections and only turn to the more involved analysis if a suitable schedule cannot be found.

By using backing-up as a fall-back heuristic we can still compute SW- and NE-closures incrementally. Then, if necessary, we can proceed to test the skipped rectangles for collisions as necessary to find a back-up schedule, in much the same way as we would incrementally test more distant rectangles in the diagonal search optimization.

7.6.7 Handling more than two robots

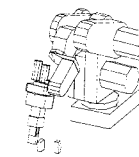




The TC diagram as we have described it will only allow the multi-robot planner to schedule two robots. The basic technique, however, is not at all limited to only two robots, and can be extended to any number of robots. This section describes two different methods to accomplish this.

Three-dimensional and higher TC diagrams It is possible to extend the concept of the TC diagram to more than two dimensions. Each axis of the multi-dimensional TC diagram represents the path planned for one of the robots. The grid structure extends trivially, though the representation of collision regions needs a little explanation.

The robots and their paths are tested pairwise for potential collision. Collision regions are generated in the two-dimensional subspaces corresponding to the pairs of robots. These collision regions extend indefinitely in all other dimensions. The physical interpretation of one of these collision regions is that whenever the pair of robots are simultaneously executing the particular path segments which potentially collide, they potentially collide *no matter where the remaining robots are in their paths.* As before, any safe schedule must remain outside all the collision regions. One can visualize a three-dimensional TC diagram as a room with bars running between opposite walls. Finding a safe schedule is done by determining a three-dimensional path between the start corner and the goal corner avoiding all the bars.



Deadlock avoidance is accomplished by the higher-dimensional analogues to the SW-closure. Once again any origin-pointing concavities must be eliminated by filling them with pseudo-collision regions. Synchronization constraints between manipulators also extend analogously.

A serious disadvantage of multi-dimensional TC diagrams is the great computation cost required to find and process the collision regions. All pairs of robot paths must be compared, so the time required to test them grows as the square of the number of robots. Some analyses of the resulting TC diagram grow exponentially with the number of robots.

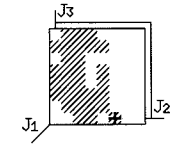
Compound manipulators Another way to handle more than two manipulators is to build "compound manipulators." For example, if we start with three six-degree-of-freedom manipulators, A, B, and C, we construct a two-dimensional TC diagram for the paths planned for A and B. We then select a safe, deadlock-free schedule for those two

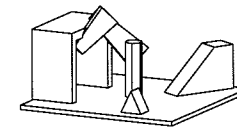
robots. By selecting such a schedule, we can, in effect, force a particular timing between the two robots. We can treat them as a single twelve-degree-of-freedom manipulator, since their joint angles are specified with respect to each other. We can then construct another two-dimensional TC diagram with the first axis representing the path for the compound manipulator and the second axis representing the path for robot C. A safe schedule in this TC diagram is then safe and deadlock-free for all three robots.

The time required to find such a safe schedule using these compound manipulators only grows linearly with the number of robots. Unfortunately, by selecting a single schedule from each of the TC diagrams along the way we are eliminating some flexibility in finding parallelism in the task. We are also forcing ourselves to require some time-coordination between the robot controllers, contrary to our original goals.

8 Conclusion

We have presented the HANDEY system as it exists today. However, like most research systems, HANDEY was not designed in its current form. Rather, it evolved into that form through a series of iterations, each motivated by limitations uncovered during experimentation with the previous version.

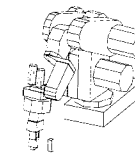




8.1 Evolution

As originally conceived, HANDEY (whose name derives from the words "hand" and "eye") was to be a much simpler system than the one presented here. We had access to a recognition system capable of locating polyhedral objects in depth maps of cluttered three-dimensional scenes [25] and a motion planning program [47] capable of planning gross motions for revolute manipulators with six or (preferably) fewer joints. The HANDEY project started out as an attempt to "connect" these two systems. In particular, it was to accomplish the following task:

1. use a laser scanner aimed at a limited, fixed area of the table to locate a modeled object among clutter,
2. grasp the object without colliding with the clutter, and
3. move the object to a specified pose.



Clearly, additional software, especially a grasp planner, was needed to complete the system.

After the object recognition system module had located the object to be moved, this elementary grasp planner was to plan the grasping operation including a short motion for the gripper from a starting point near the object to the grasp point. While the existing gross motion planner was capable of performing the latter of these operations it was felt the approach motion step more properly belonged in the grasp planner for two reasons:

1. The motions required to approach and depart an object during grasping are not gross motions. The required resolution to plan these motions would have made our gross motion planner too slow to be practical.
2. The gross motion planner had no direct way of using clutter information from the laser scanner; it required polyhedral models of the obstacles.

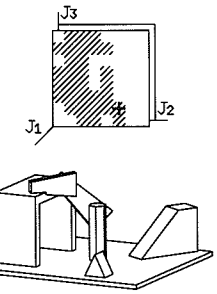
The key design decisions for the first version of the grasp planner were as follows:

- Iterate over pairs of anti-parallel object faces.
- Constrain the gripper motion to be parallel to the chosen grasp face pair.
- Project the "clutter" in the laser depth map into the plane of motion and use it during gripper motion planning.
- Select a starting point for the motion plan by using a generate-and-test strategy.
- Representing free space with a grid, find the target grasp point by shrinking mutually clear cells on the grasp faces to a point.
- Use a pseudo-potential method to plan small motions of the gripper (only).
- Verify that the gripper-based grasp plan was feasible by checking for arm collisions at pickup, robot collisions at putdown, and by checking that a continuous kinematic solution existed for the planned path. If a problem is found, try a different grasp face pair.

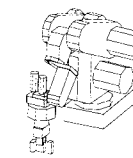
We knew that this approach was susceptible to a number of possible failures since it considered the putdown pose only in the verification step, not during the choice of the grasp, and it totally neglected kinematic constraints during choice of grasps. What we did not know was how common the failure cases would be in practice. We started out with the philosophy of trying the simple solution first.

The first live tests with the initial version of HANDEY were a revelation. Our test case was a simple L-shaped polyhedron. We found that, as long as the test involved grasping the part in a relatively uncluttered environment and moving the part parallel to the table, the system worked reasonably well. However, it was essentially impossible to construct an interesting test case that involved substantial clutter or substantial re-orientation of the part. The key problems, not surprisingly, involved precisely the constraints that we had neglected during choosing a grasp: interactions at the putdown pose, arm kinematic failures and arm collisions. This led us to adopt the obstacle backprojection strategy (described in Section 5.4.1) and to incorporate a test for kinematic feasibility and arm collisions into the inner loop of the potential path planner (described in Section 5.4.2).

The more careful treatment of kinematics and pickup/putdown interactions sharply expanded the range of pick-and-place problems that the system could handle. Nevertheless, the limited range of wrist motion of the robot combined with the limited stable states and limited grasp faces of the part combined with the limitations in the grasp planner made it essentially impossible to flip the part over between different stable states. In order to deal with this problem the regrasp planner was developed. The regrasp planner could, given an initial gripper/object affixment and a required object placement, compute a series of grasps and placements (occurring at a fixed spot on the table) which would terminate with a grasp compatible with the commanded putdown pose.



The regrasp planner was not completely self contained. Each motion that reoriented the gripper on the object required the grasp planner to plan the motion from the approach or departure point to the grasp point. A slightly different version of the grasp planner was required for this operation. The regrasp planner selected its own grasp point whereas the grasp planner had been designed also to select a grasp point. A modified version of the grasp planner was constructed that started with a given grasp point and moved the gripper away to a point which didn't leave the fingers overlapping the face. This method was used both for planning the departure of a gripper from a grasp and, by reversing the path after planning, to approach a given grasp point from a user-specified starting point. Also, the gross motion planner had to be called to plan the motions from the terminal point of one grasp to the starting point of the next. This required planning motions with the gripper close to the part and close to the table. This presented a number of challenges to the gross motion planner (see Section 8.2.2).



While the regrasp planner was able to compensate for most incompatible pickup/putdown choices the grasp planner might make, it took substantial time to plan the regrasping operations. Furthermore, the sequence of regrasping steps tended to increase the uncertainty of the object's position, increasing the likelihood that the operation would fail.

It was our perception that the excessive constraints introduced by the grasp planner forced unnecessary regrasps that motivated a complete redesign of the grasp planner along the lines described in Chapter 5.

8.2 Path planning

The initial testing also revealed limitations in the path-planning strategies, both in the potential-field approach and in the configuration-space search method.

8.2.1 Local minima

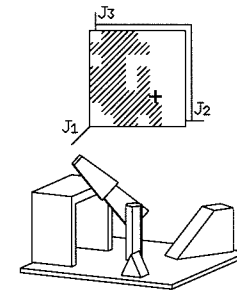
It is well known that motion planners based on potential fields are susceptible to capture by local minima in the potential. Our potential-field grasp planner for the gripper proved no exception. As long as there was a nearly straight path, with small deviations, from the start to the goal everything worked fine. But, if the straight line between the start and the goal crossed a substantial obstacle, the odds of navigating the corners were small since as the repulsive force changed directions near a corner, the attractive force and the repulsive force tended to align and cancel each other.

The fact that we were operating in a known, static environment, suggested a variation to the traditional potential method for generating the translational part of the path. This was accomplished by borrowing a technique from 2-D vision—the area of free cells connecting the starting point and grasp points were shrunken to a filament (described in Section 5.4.3). The potential-field method was then used only to compute the rotation of the gripper and small deviations from the filament path. The attraction point was moved along this path, repulsion forces generated rotations of the gripper about its reference point and small displacements. Left-Margin Movie 2 shows a path found by this planner.

8.2.2 Resolution problems

The configuration-space-based gross motion planner also ran into a number of problems in the course of experimentation. They all had to do with the problem of choosing the resolution at which to sample the configuration space. The initial version of this planner constructed a configuration space for the three-dimensional robot and parts at a fixed sampling resolution over the workspace. When the resolution was chosen to allow the robot to move with its gripper near the grasp part and the table, the computation time was unbearable.

When the resolution was chosen to lead to reasonable computation times, the robot could not approach near enough to grasp parts in cluttered environments.

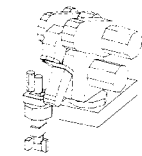


This problem led to a number of extensions to the gross-motion planner (described in Section 4.3):

- two-dimensional approximations to robot and parts,
- incremental refinement of the configuration-space map, and
- caching of previously computed maps.

The resulting planner was, on average, several order of magnitudes faster than the initial version particularly on the most common, relatively simple cases. In difficult cases, the computation time rises quickly, but this is tolerable.

8.3 Experimentation



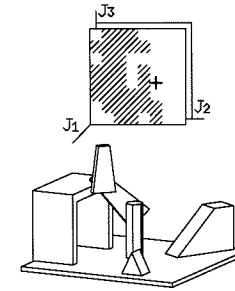
Another area where we had anticipated problems and where we were not disappointed was in calibrating our geometric and kinematic models with our experimental apparatus. Our initial experiments were with poorly modeled objects, made of wood and Styrofoam, which exacerbated the calibration difficulties. We required models to be accurate to about 1 millimeter over an area about 1 meter in radius. Our biggest problem was calibrating the table relative to the robot. Once that was done, all the parts resting on the table could be modeled reasonably well in the xy-plane, either by having the robot place them at their modeled locations or by the use of a pre-calibrated camera. Later experiments with machined objects and accurate table surfaces reduced the calibration problem to a manageable level.

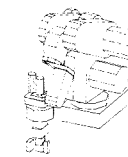
8.4 What we learned

HANDEY was not intended to be a practical system used to program commercial robots. It was intended to evaluate the current state of the art in the technology for building task-level systems. Building the system forced us to face many problems that we would have rather defined away for the sake of intellectual cleanliness. Having gone through this process, some general conclusions emerge:

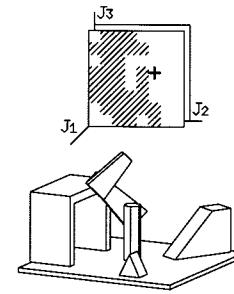
- The key difficulty in gross motion planning (Chapter 4) is not avoiding large obstacles as much as moving close enough to small ones, for example, during pickup and putdown.
- The parallel gross motion planner (Section 4.5) is exceedingly simple. Its simplicity made it very easy to debug. The pre-computed primitive maps insulated the planner from much of the problems of numerical robustness that the serial gross motion planner needs to pay close attention to. The parallel planner does suffer from the need to quantize the configuration space, and while it is very good at planning gross motions in fairly empty space, care must be taken when trying to pass close to obstacles—the quantization must be adjusted.
- The key problem in grasp planning (Chapter 5) is how to choose grasps that enable successful completion of the whole task, not just the initial grasp. This is in contrast to most of the existing literature on grasping, which focuses on some particular aspect of grasping, such as stability.
- The potential-field grasp planner (Section 5.4) proved to be more efficient than the C-space grasp planner when rotation is included. Unfortunately, it proved more difficult to incorporate other constraints beyond pure collision avoidance.
- The primary workspace of a robot is the range of positions where a full range of gripper orientations is possible. The fact that most industrial robots have an empty primary workspace is a substantial impediment to efficient task-level planning. This kinematic limitation makes it very difficult to combine planning in cartesian space with planning in the robot's joint space. It affects each of the planners that constitute HANDEY: it invalidated some simple grasps and thus forced more and longer regrasp operations (Chapter 6) than we expected. It was also very difficult to find reasonable cooperative tasks for two robots—the common workspace was very small in which both robots could manipulate objects with each robot staying reasonably out of the other's way.
- The multi-robot planner (Chapter 7) proved to be better able to achieve high levels of parallelism when the robot's paths were planned with the cooperative task in mind. Purely independent planning produced paths with many potential collisions and, thus, little possible parallelism.

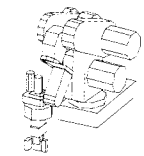
- Task-level robot programming in well-modeled workspaces is eminently practical, especially in view of the dizzying rate of growth in affordable computational power.
- HANDEY's competence in solving real three-dimensional manipulation problems, albeit limited, argues that these problems are ripe for solution and that future work in motion planning should focus on three-dimensional problems and away from the better understood and less challenging two-dimensional problems.
- Task-level robot programming in workspaces with substantial uncertainty still requires fundamental new research in planning with uncertainty and planning for the use of sensors.





Bibliography





[1] J. M. Abel, W. Holzmann, and J. M. McCarthy. On grasping planar objects with two articulated fingers. In *1985 International Conference on Robotics and Automation*, pages 576–581, St. Louis, Missouri, March 1985. IEEE Computer Society Press.

[2] R. Alami, T. Siméon, and J. P. Laumond. A geometrical approach to planning manipulation tasks: The case of discrete placements and grasps. In *Fifth International Symposium on Robotics Research*, pages 113–119, Tokyo, Japan, 1989.

[3] B. S. Baker, S. J. Fortune, and E. H. Grosse. Stable prehension with three fingers. In *17th Proc. ACM Symposium on Theory of Computing*, pages 114–120, Providence, Rhode Island, May 1985.

[4] J. Barber et al. Automatic two-fingered grip selection. In *1986 International Conference on Robotics and Automation*, pages 890–896, San Francisco, California, April 1986. IEEE Computer Society Press.

[5] M. S. Branicky and W. S. Newman. Rapid computation of configuration space obstacles. In *1990 International Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[6] D. L. Brock. Enhancing the dexterity of a robot hand using controlled slip. Master's thesis, Massachusetts Institute of Technology, Dept. of Mechanical Engr., Cambridge, Massachusetts, May 1987.

[7] R. C. Brost. Automatic grasp planning in the presence of uncertainty. *International Journal of Robotics Research*, 7(1):3–17, February 1988.

[8] P. Brou. Implementation of high level commands for robots. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1980.

[9] J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Massachusetts, 1987.

[10] J. J. Craig. Issues in the design of off-line programming systems. In R. Bolles and B. Roth, editors, *Fourth International Symposium on Robotics Research*, pages 379–389, Santa Cruz, California, August 1987. MIT Press.

[11] M. R. Cutkosky. *Robotic Grasping and Fine Manipulation.* Kluwer Academic Press, 1985.

[12] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31(3):295–353, March 1987.

[13] B. R. Donald. *Error Detection and Recovery in Robotics.* Springer-Verlag, New York, New York, 1989.

[14] R. S. B. K. M. Doshi. Task planning for "simple" assembly. Phd thesis, University of Minnesota, Minneapolis, Minnesota, October 1990.

[15] M. A. Erdmann. Using backprojections for fine-motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):19–45, Spring 1986.

[16] M. A. Erdmann. Randomization in robot tasks. In *1990 International Conference on Robotics and Automation*, pages 1744–1749, Cincinnati, Ohio, 1990. IEEE Computer Society Press.

[17] M. A. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[18] H. A. Ernst. *MH-1, A Computer-Operated Mechanical Hand.* PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Massachusetts, December 1961.

[19] J. Feldman et al. The stanford hand-eye project. In *Proc. 2nd Int. Joint Conf. Artificial Intelligence*, pages 350–358, London, England, September 1971.

[20] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *1984 International Conference on Robotics and Automation*, Atlanta, Georgia, March 1984. IEEE Computer Society Press.

[21] R. Fearing. Simplified grasping and manipulation with dextrous robot hands. In *American Control Conference*, pages 32–38, 1984.

[22] S. Fortune, G. Wilfong, and C. Yap. Coordinated motion of two robot arms. In *1986 International Conference on Robotics and Automation*, pages 1216–1223, San Francisco, California, 1986. IEEE Computer Society Press.

[23] E. Freund and H. Hoyer. Real-time pathfinding in multirobot systems including obstacle avoidance. *International Journal of Robotics Research*, 7(1), February 1988.

[24] Q. J. Ge and J. M. McCarthy. An algebraic formulation of configuration space obstacles for spatial robots. In *1990 International Conference on Robotics and Automation*, pages 1542–1547, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[25] W. E. L. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(4):469–482, July 1987.

[26] L. Guibas, L. Ramshaw, , and J. Stolfi. A kinetic framework for computational geometry. In *24th Proc. IEEE Symposium on Foundations of Computer Science*, pages 100–111, Tucson, Arizona, 1983.

[27] H. Hanafusa and H. Asada. Stable prehension of objects by robot hand with elastic fingers. In *7th International Symposium on Industrial Robots*, pages 361–368, Tokyo, Japan, October 1977. Society of Manufacturing Engineers.

[28] D. Hillis. *The Connection Machine.* MIT Press, Cambridge, Massachusetts, 1985.

[29] W. Holzmann and J. M. McCarthy. Computing the friction forces associated with a three-fingered grasp. In *1985 International Conference on Robotics and Automation*, pages 594–600, St. Louis, Missouri, April 1985. IEEE Computer Society Press.

[30] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects. *International Journal of Robotics Research*, 3(4):76–88, 1984.

[31] B. K. P. Horn. *Robot Vision.* MIT Press, Cambridge, Massachusetts, 1986.

[32] S. A. Hutchinson. A task planner for simultaneous fulfillment of operational, geometric, and uncertainty reduction goals. Technical Report 88-46, Purdue University, West Lafayette, Indiana, 1988.

[33] J. W. Jameson and L. J. Leifer. Quasi-static analysis: A method for predicting grasp stability. In *1986 International Conference on Robotics and Automation*, pages 876–883, San Francisco, California, April 1986. IEEE Computer Society Press.

[34] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, 5:72–89, 1986.

[35] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[36] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Press, Norwell, Massachusetts, 1991.

[37] C. Laugier. A program for automatic grasping of objects with a robot arm. In *11th International Symposium on Industrial Robots*, Tokyo, Japan, October 1981. Society of Manufacturing Engineers.

[38] C. Laugier and J. Pertin-Troccaz. Automatic robot programming: the grasp planner. In *International Symposium on Advanced Software in Robotics*, Liège, Belgium, 1983.

[39] C. Laugier and J. Pertin-Troccaz. Sharp: A system for automatic programming of manipulation robots. In *Third International Symposium on Robotics Research*, pages 125–132, Gouvieux-Chantilly, France, 1985.

[40] J. P. Laumond and R. Alami. A geometrical approach to planning manipulation tasks: The case of a circular robot and movable circular object amidst polygonal obstacles. Technical Report 88314, LAAS/CNRS, Toulouse, France, 1988.

[41] J. P. Laumond and R. Alami. A geometrical approach to planning manipulation tasks in robotics. In *Proc. 1st Canadian Conference on Computational Geometry*, Montréal, Canada, 1989.

[42] L. I. Lieberman and M. A. Wesley. AUTOPASS: An automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research & Development*, 21(4):321–333, January 1977.

[43] T. Lozano-Pérez. The design of a mechanical assembly system. Technical Report AI-TR-397, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, December 1976.

[44] T. Lozano-Pérez. Automatic planning of manipulator transfer movement. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(10):681–698, October 1981.

[45] T. Lozano-Pérez. Robot programming. *Proc. IEEE*, 71(7):821–841, July 1983.

[46] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983.

[47] T. Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. *IEEE Journal on Robotics and Automation*, RA-3(3):224–238, June 1987.

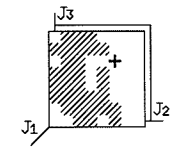
[48] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell, P. Tournassoud, and P. Lanusse. Handey: A task-level robot system. In *Fourth International Symposium on Robotics Research*, pages 29–36, Santa Cruz, California, August 1987.

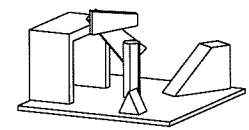
[49] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, Spring 1984.

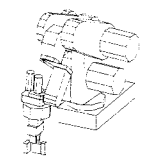
[50] T. Lozano-Pérez and R. H. Taylor. Geometric issues in planning robot tasks. In M. Brady, editor, *Robotics Science.* MIT Press, Cambridge, Massachusetts, 1988.

[51] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[52] M. T. Mason. Compliant motion. In J. M. Brady, J. M. Hollerbach, T. Johnson, T. Lozano-Pérez, and M. T. Mason, editors, *Robot Motion.* MIT Press, Cambridge, Massachusetts, 1983.







[53] M. T. Mason. Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research*, 5(3):53–71, Fall 1986.

[54] E. Mazer. LM-geo: Geometric programming of assembly robots. In A. Danthine and M. Géradin, editors, *Advanced Software in Robotics*, pages 99–110. North-Holland, Amsterdam, 1983.

[55] J. M. McCarthy and R. M. C. Bodduluri. A bibliography on robot kinematics, workspace analysis, and path planning. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *Robotics Review 1*. MIT Press, Cambridge, Massachusetts, 1989.

[56] W. S. Newman. *High-Speed Robot Control in Complex Environments*. PhD thesis, Massachusetts Institute of Technology, Dept. of Mechanical Engr., 1987.

[57] V. D. Nguyen. The synthesis of stable force closure grasps. Technical Report AI-TR-905, Massachusetts Institute of Technology, Cambridge, Massachusetts, July 1986.

[58] V. D. Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3):3–16, 1988.

[59] V. D. Nguyen. Constructing stable grasps. *International Journal of Robotics Research*, 8(1):26–37, February 1989.

[60] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, California, 1980.

[61] T. Okada. Computer control of multijointed finger system for precise object handling. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-12(3):289–299, May 1982.

[62] R. P. Paul. Modelling, trajectory calculation, and servoing of a computer controlled arm. PhD thesis AIM-177, Stanford University, Artificial Intelligence Lab., Stanford, California, November 1973.

[63] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Massachusetts, 1981.

[64] J. Pertin-Troccaz. On-line automatic robot programming: a case study in grasping. In *1987 International Conference on Robotics and Automation*, pages 1292–1297, Raleigh, North Carolina, 1987. IEEE Computer Society Press.

[65] J. Pertin-Troccaz. Grasping: A state of the art. In O. Khatib, J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review 1*, pages 71–98. MIT Press, Cambridge, Massachusetts, 1989.

[66] D. L. Pieper. The kinematics of manipulators under computer control. Technical Report AI 72, Stanford University, Computer Science Dept., Stanford, California, October 1968.

[67] R. J. Popplestone, A. P. Ambler, and I. M. Bellos. Rapt, a language for describing assemblies. *Industrial Robotics*, 5(3):131–137, 1978.

[68] R. J. Popplestone, A. P. Ambler, and I. M. Bellos. An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14(1):79–107, 1980.

[69] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, New York, New York, 1985.

[70] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *26th Proc. IEEE Symposium on Foundations of Computer Science*, pages 144–154, Portland, Oregon, 1985.

[71] J. T. Schwartz and M. Sharir. On the piano mover's problem III: Coordinating the motion of several independent bodies: the special case of circular bodies amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983.

[72] M. Sharir. Algorithmic motion planning in robotics. *IEEE Computer*, 22(3), March 1989.

[73] R. H. Taylor. A synthesis of manipulator control programs from task-level specifications. PhD thesis AIM-282, Stanford University, Artificial Intelligence Lab., Stanford, California, 1976.

[74] R. H. Taylor. Review of: MH-1, a computer-operated mechanical hand. In O. Khatib, J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review 1*. MIT Press, Cambridge, Massachusetts, 1989.

[75] P. Tournassoud. A strategy for obstacle avoidance and its application to multi-robot systems. *IEEE Journal on Robotics and Automation*, pages 1224–1229, 1986.

[76] P. Violero, I. Mazon, and M. Taïx. Automatic planning of a grasp for a "pick and place" action. In *1990 International Conference on Robotics and Automation*, pages 870–876, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[77] M. A. Wesley, T. Lozano-Pérez, L. I. Lieberman, M. A. Lavin, and D. D. Grossman. A geometric modeling system for automated mechanical assembly. *IBM Journal of Research & Development*, 24(1):64–74, January 1980.

[78] S. H. Whitesides. Computational geometry and motion planning. In G. T. Toussaint, editor, *Computational Geometry*, pages 377–427. North-Holland, Amsterdam, 1985.

[79] D. E. Whitney. Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, 6(1):3–14, Spring 1987.

[80] G. Wilfong. Motion planning in the presence of movable obstacles. In *Proc. 4th ACM Symposium on Computational Geometry*, pages 279–288, Urbana-Champaign, Illinois, 1988.

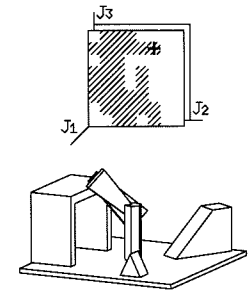
[81] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, Los Altos, California, 1988.

[82] M. Wingham. Planning how to grasp objects in a cluttered environment. Master's thesis, University of Edinburgh, Edinburgh, Scotland, 1977.

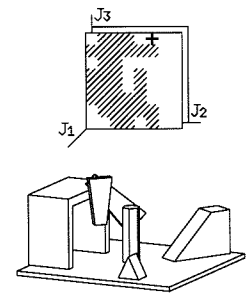
[83] J. D. Wolter, R. A. Volz, and A. C. Woo. Automatic generation of gripping positions. Technical report, University of Michigan, Ann Arbor, Michigan, 1984.

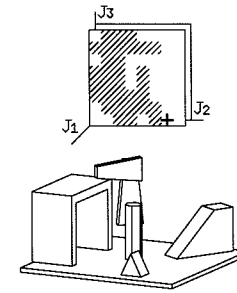
[84] M. Z. Yannanakis, C. H. Papadimitiou, and H. T. Kung. Locking policies: Safety and freedom from deadlock. In *20th Proc. IEEE Symposium on Foundations of Computer Science*, pages 286–297, San Juan, Puerto Rico, 1979.

[85] C. K. Yap. Algorithmic motion planning. In J. T. Schwartz and C. K. Yap, editors, *Advances in Robotics: Volume 1*. Lawrence Erlbaum Associates, 1987.



Index



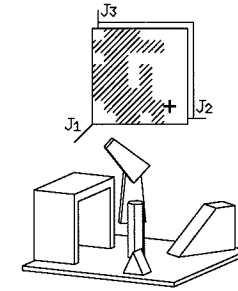


U

V

W

Y



Artificial Intelligence

Patrick Henry Winston, founding editor

J. Michael Brady, Daniel G. Bobrow, and Randall Davis, current editors

Artificial Intelligence: An MIT Perspective, Volume I: Expert Problem Solving, Natural Language Understanding, Intelligent Computer Coaches, Representation and Learning, edited by Patrick Henry Winston and Richard Henry Brown, 1979

Artificial Intelligence: An MIT Perspective, Volume II: Understanding Vision, Manipulation, Computer Design, Symbol Manipulation, edited by Patrick Henry Winston and Richard Henry Brown, 1979

NETL: A System for Representing and Using Real-World Knowledge, Scott Fahlman, 1979

The Interpretation of Visual Motion, by Shimon Ullman, 1979

A Theory of Syntactic Recognition for Natural Language, Mitchell P. Marcus, 1980

Turtle Geometry: The Computer as a Medium for Exploring Mathematics, Harold Abelson and Andrea di Sessa, 1981

From Images to Surfaces: A Computational Study of the Human Visual System, William Eric Leifur Grimson, 1981

Robot Manipulators: Mathematics, Programming, and Control, Richard P. Paul, 1981

Computational Models of Discourse, edited by Michael Brady and Robert C. Berwick, 1982

Robot Motion: Planning and Control, edited by Michael Brady, John M. Hollerbach, Timothy Johnson, Tomás Lozano-Pérez, and Matthew T. Mason, 1982

In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension, Michael G. Dyer, 1983

Robotic Research: The First International Symposium, edited by Hideo Hanafusa and Hirochika Inoue, 1985

Robot Hands and the Mechanics of Manipulation, Matthew T. Mason and J. Kenneth Salisbury, Jr., 1985

The Acquisition of Syntactic Knowledge, Robert C. Berwick, 1985

The Connection Machine, W. Daniel Hillis, 1985

Legged Robots that Balance, Marc H. Raibert, 1986

Robotics Research: The Third International Symposium, edited by O.D. Faugeras and Georges Giralt, 1986

Machine Interpretation of Line Drawings, Kokichi Sugihara, 1986

ACTORS: A Model of Concurrent Computation in Distributed Systems, Gul A. Agha, 1986

Knowledge-Based Tutoring: The GUIDON Program, William Clancey, 1987

AI in the 1980s and Beyond: An MIT Survey, edited by W. Eric L. Grimson and Ramesh S. Patil, 1987

Visual Reconstruction, Andrew Blake and Andrew Zisserman, 1987

Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence, Yoav Shoham, 1988

Model-Based Control of a Robot Manipulator, Chae H. An, Christopher G. Atkeson, and John M. Hollerbach, 1988

A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control, Russell L. Andersson, 1988

Robotics Research: The Fourth International Symposium, edited by Robert C. Bolles and Bernard Roth, 1988

The Paralation Model: Architecture-Independent Parallel Programming, Gary Sabot, 1988

Concurrent System for Knowledge Processing: An Actor Perspective, edited by Carl Hewitt and Gul Agha, 1989

Automated Deduction in Nonclassical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics, Lincoln Wallen, 1989

Shape from Shading, edited by Berthold K.P. Horn and Michael J. Brooks, 1989

Ontic: A Knowledge Representation System for Mathematics, David A. McAllester, 1989

Solid Shape, Jan J. Koenderink, 1990

Expert Systems: Human Issues, edited by Dianne Berry and Anna Hart, 1990

Artificial Intelligence: Concepts and Applications, edited by A. R. Mirzai, 1990

Robotics Research: The Fifth International Symposium, edited by Hirofumi Miura and Suguru Arimoto, 1990

Theories of Comparative Analysis, Daniel S. Weld, 1990

Artificial Intelligence at MIT: Expanding Frontiers, edited by Patrick Henry Winston and Sarah Alexandra Shellard, 1990

Vector Models for Data-Parallel Computing, Guy E. Blelloch, 1990

Experiments in the Machine Interpretation of Visual Motion, David W. Murray and Bernard F. Buxton, 1990

Object Recognition by Computer: The Role of Geometric Constraints, W. Eric L. Grimson, 1990

Representing and Reasoning With Probabilistic Knowledge: A Logical Approach to Probabilities, Fahiem Bacchus, 1990

3D Model Recognition from Stereoscopic Cues, edited by John E.W. Mayhew and John P. Frisby, 1991

Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception, Nicholas Ayache, 1991

Truth and Modality for Knowledge Representation, Raymond Turner, 1991

Made-Up Minds: A Constructivist Approach to Artificial Intelligence, Gary L. Drescher, 1991

Vision, Instruction, and Action, David Chapman, 1991

Do the Right Thing: Studies in Limited Rationality, Stuart Russell and Eric Wefeld, 1991

KAM: A System for Intelligently Guiding Numerical Experimentation by Computer, Kenneth Man-Kam Yip, 1991

Solving Geometric Constraint Systems: A Case Study in Kinematics, Glenn A. Kramer, 1992

Geometric Invariants in Computer Vision, edited by Joseph Mundy and Andrew Zisserman, 1992

HANDEY: A Robot Task Planner, Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, and Patrick A. O'Donnell, 1992

The MIT Press, with Peter Denning as general consulting editor, publishes computer science books in the following series:

ACL-MIT Press Series in Natural Language Processing
Aravind K. Joshi, Karen Sparck Jones, and Mark Y. Liberman, editors

ACM Doctoral Dissertation Award and Distinguished Dissertation Series

Artificial Intelligence
Patrick Winston, founding editor
J. Michael Brady, Daniel G. Bobrow, and Randall Davis, editors

Charles Babbage Institute Reprint Series for the History of Computing
Martin Campbell-Kelly, editor

Computer Systems
Herb Schwetman, editor

Explorations with Logo
E. Paul Goldenberg, editor

Foundations of Computing
Michael Garey and Albert Meyer, editors

History of Computing
I. Bernard Cohen and William Aspray, editors

Logic Programming
Ehud Shapiro, editor; Fernando Pereira, Koichi Furukawa, Jean-Louis Lassez, and David H. D. Warren, associate editors

The MIT Press Electrical Engineering and Computer Science Series

Research Monographs in Parallel and Distributed Processing
Christopher Jesshope and David Klappholz, editors

Scientific and Engineering Computation
Janusz Kowalik, editor

Technical Communication and Information Systems
Ed Barrett, editor