ON COMPUTING MULTI-ARM MANIPULATION TRAJECTORIES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Yoshihito Koga
November 1994

Abstract

This dissertation considers the manipulation task planning problem of automatically generating the trajectories for several cooperating robot arms to manipulate a movable object to a goal location among obstacles. The planner must reason that the robots may need to change their grasp of the object to complete the task, for example, by passing it from one arm to another. Furthermore, the computed velocities and accelerations of the arms must satisfy the limits of the actuators. Past work strongly suggests that solving this problem in a rigorous fashion is intractable.

We address this problem in a practical two-phase approach. In step one, using a heuristic we compute a collision-free path for the robots and the movable object. For the case of multiple robot arms with many degrees of freedom, this step may fail to find the desired path even though it exists. Despite this limitation, experimental results of the implemented planner (for solving step one) show that it is efficient and reliable; for example, the planner is able to find complex manipulation motions for a system with seventy eight degrees of freedom. In step two, we then find the time-parameterization of the path such that the dynamic constraints on the robot are satisfied. In fact, we find the time-optimal solution for the given path. We show simulation results for various complex examples.

Acknowledgements

I wish to thank my advisor Professor Jean-Claude Latombe for his skillful guidance and encouragement during the course of the research. I was extremely fortunate to have him for an advisor.

I would also like to thank Professor Oussama Khatib and Professor Mark Cutkosky for serving on my reading committee and for the many helpful discussions.

Thank you to Koichi Kondo, James Kuffner and Jonathan Norton for putting up with me while collaborating on the human arm manipulation project. I am grateful for the opportunity to have worked with them. Much thanks also to Tsai-Yen Li for the many, many hours of help he gave me in debugging ideas on manipulation planning. I am also indebted to Sean Quinlan for the use of his fast collision-detection routine, and to Jerome Barraquand for guidance during the early stages of the work.

The Computer Science Robotics Laboratory was a fantastic place to be in with all its friendly people. Thank you everyone for all your kindness. Thanks go to Mark Yim, Lydia Kavraki, and Tsai-Yen Li for their help and encouragement during the writing stage of the thesis.

Special thanks to Peter Chow and Margaret Abe for being such incredibly cool people.

Finally, I'd like to thank Mom, Dad and Midori for their love, encouragement, and help in the pursuit of my goals.

Financial support for this work was provided by a Canadian NSERC fellowship and ONR contract N00014-92-J-1809.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem and Motivation

Robots today are doing useful work. In the factories, robots on the assembly line efficiently execute repetitive pick-and-place operations. In the hospitals, robots assist in difficult surgical procedures. In outer space, robots perform the dangerous manipulation task of capturing satellites. These completely different tasks and environments are just a few examples of where robots are making a contribution. The next step is to make them easier to use. Indeed, as robot systems become more flexible in dealing with greater varieties of tasks, they also become more difficult to use.

Robot Motion Planning is the field within robotics aimed at solving this problem. In its fundamental form, robot motion planning is the automatic computation of robot motions to achieve some goal arrangement of the environment. This could be the *piano movers' problem* of finding a collision-free path for a robot to move from one configuration to another. A more sophisticated problem would be the *manipulation planning problem* of automatically computating a *series* of robot motions to manipulate an object to a specified goal location. Regardless of the different tasks involved, the goal of robot motion planning is to make robots autonomous. Thus ultimately, a vague high-level command such as "insert the engine" would result in say two robot arms automatically moving to grab the engine, placing it into the car, and then fixing it onto the engine block.

The components needed to facilitate this high-level interaction with robots, such as sensing and planning, are major areas of research in robotics today. This dissertation presents our efforts towards solving one piece of this bigger puzzle - given a goal location for a movable object, automatically compute the cooperative motion for multiple robot arms to manipulate the object to its specified goal. The motion must be collision-free, and the velocities and accelerations must be within the capabilities of the robot actuators.

An example of the problem is illustrated in Fig. 1.1. Given a model of the robots, obstacles, and the movable object, and the high-level task specification "move the L-shaped object to the other side of the workspace" we wish to develop a planner that automatically generates the arm motions to complete the task. Snapshots (a) to (l) in Fig. 1.1 is one solution to the task. This is a very intricate problem involving grasping, regrasping, and cooperation between robots. Specifically, many different questions such as:

- how should the arm(s) intially grasp the object;
- which arm(s) at any one time should be responsible for manipulating the object;
- when and how should the arm(s) regrasp;
- how should the arm(s) cooperate to ensure the delivery of the object to its specified goal;

need to be addressed. We formalize and study these difficult questions from which we develop an implemented algorithm that solves many complicated instances of this multi-arm problem. The solution given in Fig. 1.1 was generated by our planner.

We focus on multi-arm systems because of their inherent flexibility. In contrast to single-arm systems, multi-arm systems can be significantly more efficient by performing motions simultaneously. They can also accomplish a greater variety of tasks. For example, in addition to the arms working independently of each other, they can cooperate to manipulate heavy and/or bulky objects by sharing the load for fast and responsive motions. One can also increase the reachable space of the movable objects by having the robots pass the object from one arm to another. With the advent of

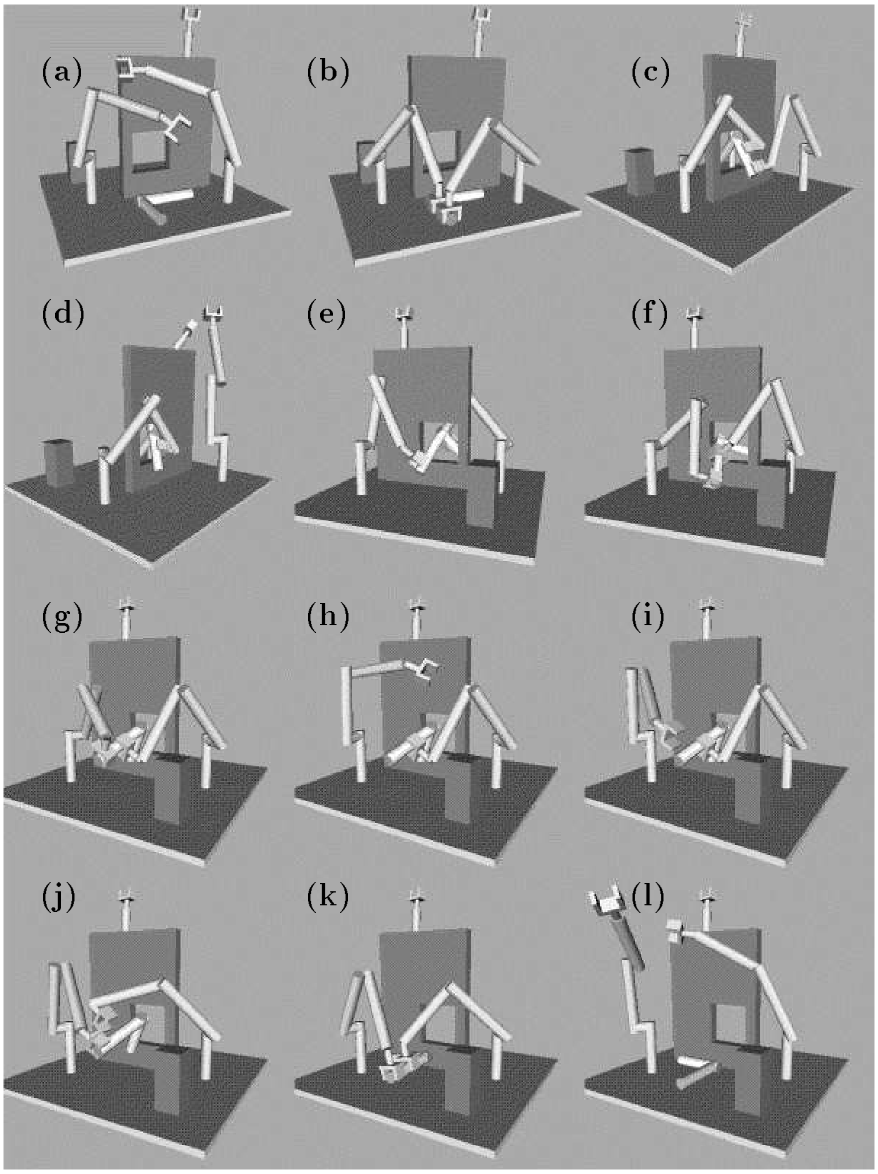


Figure 1.1: An example manipulation motion.

new control techniques for such multi-arm systems, there already exist real robots in need of a planner to simplify their use [Khatib, 1988][Schneider, 1992].

A motion planner for multiple robot arms has other applications. By replacing the robot models with those of the human arms, the planner could automatically generate simulations of human arm manipulation motions. Since human figures often play an integral role in computer animation, their motions could be easily generated from task-level specifications by using the planner. The planner could also be useful in ergonomics. Since most products are utilized, assembled, maintained, and repaired by humans, and require for most cases some action by the human arms, one could evaluate the design of a product in terms of its usability by viewing the simulation of arm motions generated by the planner. This would reduce the number of mock-up models needed to come up with the final design.

1.2 Computational Complexity and Completeness

Consider the problem of moving a robot from one configuration to another in a collision-free manner. The fastest known complete algorithm for solving this problem is one with time complexity that is singly-exponential in the number of degrees of freedom of the robot [Canny, 1988]. This algorithm is complete in the sense that given a description of the robot and the environment as semi-algebraic sets, it finds a collision-free path connecting the intial and goal configuration of the robot *when it exists*, and reports failure *when it does not*. The algorithm is thorough but impractical for any robot with many degrees of freedom. Although no proof exists to show that the problem is intractable, the work of Canny [Canny, 1988], the establishment of the PSPACE-hardness of the problem [Reif, 1979], and other related theoretical work as cited in Chapter 1 of Latombe's book [Latombe, 1991] strongly suggests that this is the case.

Consider now our problem of moving multiple robots through a series of steps to manipulate an object to a specified goal location. If the best algorithm to find a collision-free path for a single robot has exponential-time-complexity, then including additional robots and the added difficulty of having to find a series of robot motions to manipulate the object (this would involve grasping, ungrasping, and possibly finding intermediate placements for the object for regrasping) leaves us with a problem that is at least as hard.

Since completeness implies the algorithm is inapplicable to any realistic robot system with many degrees of freedom, we trade-off completeness for an algorithm that efficiently and reliably solves a majority of realistic manipulation tasks involving multiple arms. The *practical* planner we seek must then have the property that it reliably and efficiently deal with robot systems with many degrees of freedom, while including regrasping operations when they are required to complete the task, and ensuring that the velocities and accelerations of the robots are within the capabilities of the actuators. Although not necessary for practicality, we also include the time-optimal aspect.

Existing methods to determine the time-optimal trajectory between two states of a robot with many degrees of freedom are notoriously slow to converge on a solution [Slattery, 1991]. This is without the obstacle avoidance problem. Based on this result, it seems unlikely to find a practical algorithm that in one step computes time-optimal manipulation trajectories. This leaves us with a two-step approach; find the collision-free paths and then time parameterize them while satisfying the dynamic constraints and minimizing the travel time (for the given path). We refer to the first step as the **manipulation planning problem** and the second step as the **time parameterization problem**.

Fortunately this two-step approach is feasible. In the past few years much progress has been made in the development of practical motion planners for high degree of freedom robots [Barraquand and Latombe, 1991][Kondo, 1991][Kavraki and Latombe, 1994][Overmars and Svestka, 1994][Gupta and Zhu, 1994]. These planners lack the thoroughness of a complete algorithm, but for a majority of problems they find a solution in a short amount of time. Given the success of these approaches, the core problem addressed in this work is the design and implementation of a practical planner to find the series of multi-arm paths to solve realistic manipulation tasks.

To time parameterize the computed paths, we use the well established algorithm of [Bobrow et al., 1985] and [Shin and Mackay, 1985]. The algorithm has linear-time-complexity in the number of discrete points making up the path, resulting in fast computation.

1.3 Related Work

The related work can be broken into two parts; manipulation planning, which considers the spatial aspect, and time-optimal control, which considers the temporal aspect.

1.3.1 Manipulation Planning

Path planning for one robot among fixed obstacles and various extensions of this basic problem have been actively studied during the past two decades [Latombe, 1991]. However, research strictly addressing manipulation planning is fairly recent.

The first paper to tackle this problem is by Wilfong [Wilfong, 1988]. It considers a single-body robot translating in a 2D workspace with multiple movable objects. The robot, movable objects, and obstacles are modeled as convex polygons. The robot "grasps" an object by making one of its edges coincide with an edge of the object. This definition of "grasping" extends to several movable objects. Wilfong shows that planning a manipulation path to bring the movable objects to their specified goal locations is PSPACE-hard. When there is a single movable object, he proposes a complete algorithm that runs in $O(n^3 \log^2 n)$ time, where n is the total number of vertices of all the objects in the environment. Laumond and Alami [Laumond and Alami, 1988] propose an $O(n^4)$ algorithm to solve a similar problem where the robot and the movable object are both discs and the obstacles are polygonal.

Alami, Siméon and Laumond [Alami et al., 1990] describe a manipulation planner for one robot and several movable objects. Both the number of legal grasps of each object (positions of the robot relative to the object) and the number of legal placements of the movable objects are finite. The method was implemented for two simple robots: a translating polygon [Alami et al., 1990] and a three-revolute-joint planar arm [Laumond and Alami, 1989]. A theoretical study of the more general case where the set of legal grasps and placements of the movable objects are continuous sets is presented in [Laumond and Alami, 1989]. In our practical approach we concentrate on planning motions for multiple robot arms and a single movable object, as opposed to multiple movable objects and a single robot.

Ferbach and Barraquand [Ferbach and Barraquand, 1993], introduce a practical approach to the multi-arm manipulation planning problem using a variational technique based on dynamic programming. They first solve the problem assuming the movable object can be moved without the arms grasping it. This path is then varied to force the robots to actually grab the object in order for it to move. Though many interesting problems can be solved, their system is not fully automatic and requires as input intermediate goal locations for the movable object.

Lynch addresses the problem of planning pushing paths [Lynch, 1993]. He establishes the conditions under which the contact between the robot and the movable object is stable, given the friction coefficients and the center of friction between the movable object and its supporting surface. These conditions yield nonholonomic constraints on the the motion of the robot. One could view this pushing motion as a manipulation task. However, Lynch does not consider the regrasping operations.

Regrasping is a vital component in manipulation tasks. Tournassoud, Lozano-Pérez, and Mazer [Tournassoud et al., 1987] specifically address this problem. They describe a method for planning a sequence of regrasp operations by a single arm to change an initial grasp into a goal grasp. At every regrasp, the object is temporarily placed on a horizontal table in a stable position selected by the planner.

The work on regrasping presented in [Tournassoud et al., 1987] is part of an integrated manipulation system, HANDEY, described in [Lozano-Pérez et al., 1987]. This system controls a PUMA arm which builds an assembly in a 3D workspace. It integrates vision, path planning, grasp planning, and motion control. While it embeds a solution to many issues not considered in this thesis, it does not address the problem of planning cooperative robot motions to accomplish manipulation tasks.

Planning coordinated paths for multiple robots without movable objects, is studied in [O'Donnel and Lozano-Pérez, 1989]. Their approach consists of two steps. The collision-free paths for the multiple robots are computed independently of each other, and then coordinated such that the robots avoid collision with one another. The method is based on a scheduling technique for dealing with limited resources. In our work, we do not restrict the problem by decoupling the planning of the multiple arms.

Grasp planning is potentially an important component of manipulation planning. One must decide how to position the fingers of the gripper on the object such that geometric and physical constraints are satisfied. An example would be ensuring that the grasp is stable, that is any slight displacement of the object tends to return the object to its original position within the grasp. There exist implemented grasp planners for parallel-jaw grippers grasping polyhedral objects [Laugier and Pertin, 1983][Tournassoud et. al, 1987]. See [Pertin-Troccaz, 1989] for a broad survey of publications related to grasping.

Another potentially important aspect of manipulation planning is determining the stable placements of the object for regrasping actions. These would be intermediate placements of the object where the robots can completely ungrasp for the purpose of regrasping. Using a simplified model of friction, Boneschanscher et al. [Boneschanscher et al., 1988] propose an algorithm that tests the stability of an arrangement of polyhedra on a table in the six-dimensional space of translations and rotations.

1.3.2 Time-Optimal Control

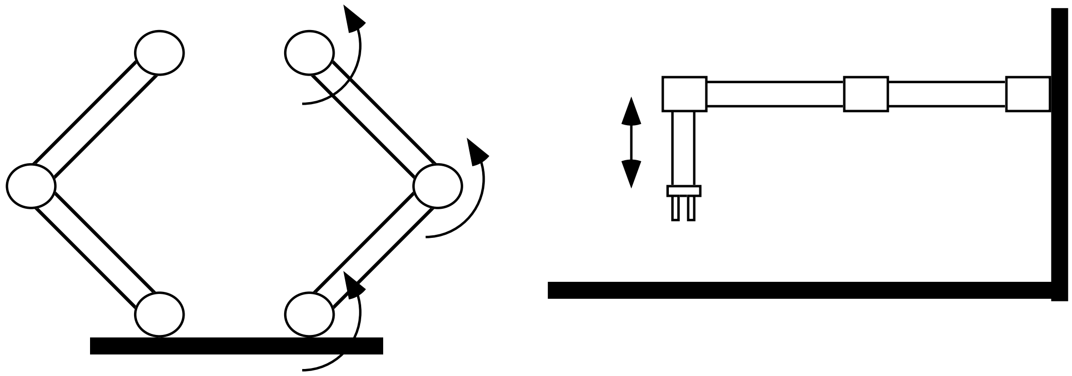
The time-optimal control problem for robots has attracted considerable interest. There are two approaches for tackling this *two point boundary value problem* (TPBVP). The first approach involves finding in one step the optimal path shape and the corresponding control history that yields the global minimum travel time. The second approach is to add a geometric path constraint to the TPBVP, thus restricting the problem to finding the optimal control history that yields the minimum travel time for the *specified* path.

The first to consider this problem under the general approach are Khan and Roth [Khan and Roth, 1971]. They find the near time-optimal solution of a three-degrees-of-freedom open-kinematic-chain manipulator. However, they simplify the problem by linearizing the equations of motion of the robot. A solution for a two-link manipulator without linearizing the equations of motion, is given by Meier and Bryson [Meier and Bryson, 1987]. Unfortunately, for more realistic robot systems with high degrees of freedom their method becomes impractical. An algorithm for a closed chain robot has been developed by Slattery [Slattery, 1991]. Unfortunately, even for a relatively simple robot system the method is extremely slow in converging to the solution. None of these methods consider obstacle avoidance.

Jacobs et al. [Jacobs et al., 1989] present an algorithm that guarantees bounds on the closeness of an approximation to the global-time-optimal trajectory that includes obstacle avoidance. However, the computational complexity of this algorithm is such that it is impractical for realistic systems. Some other papers on the theoretical aspects of this problem with obstacle avoidance are [Canny et al., 1988] and [Donald and Xavier, 1989].

By introducing path constraints (prescribing the robot path) the problem is greatly simplified [Bobrow et al., 1985][Shin and Mackay, 1985]. This is due to the dramatic reduction of the search space for the optimal solution. In particular, the algorithm developed by Bobrow et al. [Bobrow et al., 1985] solves this path constrained problem for an open-kinematic-chain robot in very short time. In addition, by using the output of some path planner as the prescribed path, obstacle avoidance is satisfied. Furthermore, by perturbing this path shape and finding the minimum travel time for each new shape, a path that achieves a local minimum of the travel time while avoiding obstacles can be found [Shiller and Dubowsky, 1991][Bobrow, 1988].

Others have modified Bobrow's original algorithm to find the path-constrained time-optimal motion for an open-kinematic-chain robot carrying a payload, and for a closed-kinematic-chain robot system. Shiller and Dubowsky [Shiller and Dubowsky, 1989] find the time-optimal motion of an open-kinematic-chain robot subject to payload constraints. Because the inertial effects on the payload may exceed the capacity of the gripper to hold onto the payload, Shiller and Dubowsky consider a *no-slip* constraint for the payload. They model the friction interaction between the gripper and the payload, however they only consider the translational slipping case. McCarthy



(a) Top View **(b) Side View**

Figure 1.2: The robots for the planar workspace.

and Bobrow generalize Bobrow's original algorithm for dealing with closed-kinematic-chain robot systems [McCarthy and Bobrow, 1992]. However, this algorithm does not consider the control strategy used by the real robots. In the actual execution of the optimal trajectory, the multi-arm controller may partition the tracking effort to the arms in such a way that one or more actuator exceeds their capacity.

1.4 Contributions

We present a practical two-step method to automatically compute path-constrained time-optimal multi-arm manipulation trajectories. Whereas much of the previous work in manipulation planning and kinodynamics is theoretical in nature, the focus of this work is on developing an effective approach to computing the robot motions to complete manipulation tasks of a complexity comparable to that of tasks encountered in manufacturing and construction work (e.g., assembling, welding and/or riveting the body of a car or the fuselage of a plane, assembling truss structures). This dissertation describes the progress towards this end.

Manipulation Planning in a 2D Workspace To gain insight into what is needed to develop a practical planner we carefully study a simple example. We introduce an implemented manipulation planner for a specific case of two SCARA[1] type robot arms working in an environment where the obstacles are cylindrical bodies of infinite height. Both arms have four degrees of freedom; shoulder, elbow and wrist revolute joints, and a prismatic joint to move the end effector up and down (see Fig. 1.2). By the nature of the robots and the environment, the movable object moves essentially in a plane, that is the workspace of the arms is two-dimensional. The input to the planner is a model of the obstacles in the environment, the movable object and the robot arms, and a manipulation task specified by the goal location for the object and the arms. The search space is discretized into a fine grid, and *for the given discretization*, this algorithm is guaranteed to find a series of collision-free paths to manipulate the movable object to the desired goal location if it exists and report failure otherwise. In the event of a failure, there may be a finer resolution for which a solution exists. We call this a resolution-complete algorithm.

Manipulation Planning in a 3D Workspace The methods for the 2D workspace are extended for the 3D workspace with multiple manipulator arms. We introduce an implemented *practical* manipulation planner for a three-dimensional workspace. This planner is capable of dealing with multiple robot arms of various kinematics. The only restriction on the method is that the arms must have some inverse kinematics solution. The algorithm cannot guarantee that a solution will be found if it exists and report failure if it does not exist. To keep the planner from running indefinitely, it is terminated when a preset time limit is reached. Despite this limitation, various experiments demonstrate the efficiency and reliability of the approach in finding solutions for realistic tasks. In particular, we compute the manipulation motions for a system of three general 6R manipulators, a system of three PUMA robots, and the arms of a human figure.

[1]SCARA stands for selectively compliant assembly robot arm.

Time-Optimal Control For Multi-Arm Systems Finally, a time-optimal control scheme is presented for parameterizing the collision-free paths found by these planners. We utilize an existing path-constrained optimal-control algorithm. However, we introduce additional constraints to deal with the variety of dynamic systems arising in the manipulation motions. Specifically, we add a no-slip constraint to ensure that the payload remains rigidly fixed to the robots. Also, for the multi-grasp case (i.e., multiple arms grabbing the same object), we add a control consistency constraint to ensure that the optimization deals with the redundant actuation in the same fashion as the actual multi-arm controller.

1.5 Outline

The dissertation is organized as follows:

In Chapter 2 the geometric aspect of the multi-arm manipulation problem is formally defined. This is done within the framework of configuration space.

In Chapter 3 a presentation of a resolution-complete manipulation planner for a two-dimensional workspace is given. By adding a few restrictions to the problem we describe a novel approach to solving the multi-arm manipulation problem. This chapter serves to develop a framework for dealing with the more complex problem in three-dimensional workspaces.

In Chapter 4 the multi-arm manipulation planner for three-dimensional workspaces is presented. Again, restrictions are added to the general problem to make it more manageable. However, the result is still a system capable of dealing with many manipulation tasks found in manufacturing and construction. A variety of example paths generated by our planner are given.

In Chapter 5 an algorithm for computing time-optimal motions of the robot arms constrained to the previously computed manipulation paths is described. The scheme is based on the algorithm proposed independently by Bobrow et al. [Bobrow et al., 1985] and by Shin and Mackay [Shin and Mackay, 1985].

Finally, in Chapter 6 we discuss directions for future research.

Chapter 2

Problem Statement

We address the problem of automatically computing the trajectories of multiple robot arms to manipulate an object to a specified location. We approach this problem by splitting it into two steps; a first step that deals with the spatial aspect, and a second step that deals with the temporal aspect.

By artificially breaking the problem into two steps, we limit the scope of tasks that can be considered; the path found in step one must admit a time parameterization (computed in step two) that satisfy the dynamic constraints on the robots. These are tasks where a quasi-static motion is sufficient for its completion, that is the robots can be slowed down to a crawling speed and still traverse the whole path. Consequently, step one is restricted to consider only those configurations that yield sufficient torque to admit quasi-static motion. Tasks that require the dynamic effects of the system to be utilized for its completion can not be considered. For example, lifting a heavy object to some high platform that first requires swinging the object back and forth to generate sufficient momentum can not be solved by our two step approach.

In this chapter we make precise these notions and present the multi-arm manipulation planning problem using the configuration space formalization. This is the problem statement for the first step of our two-phase approach for computing manipulation trajectories. Much of this chapter is an extension of ideas presented in [Alami et al., 1990] and Chapter 11 of [Latombe, 1991]. Our presentation is general except for the simplification of considering only a single movable object.

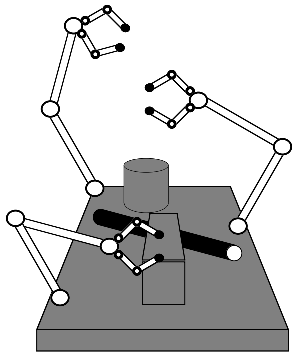


Figure 2.1: A multi-arm robotic workcell.

2.1 Configuration Space

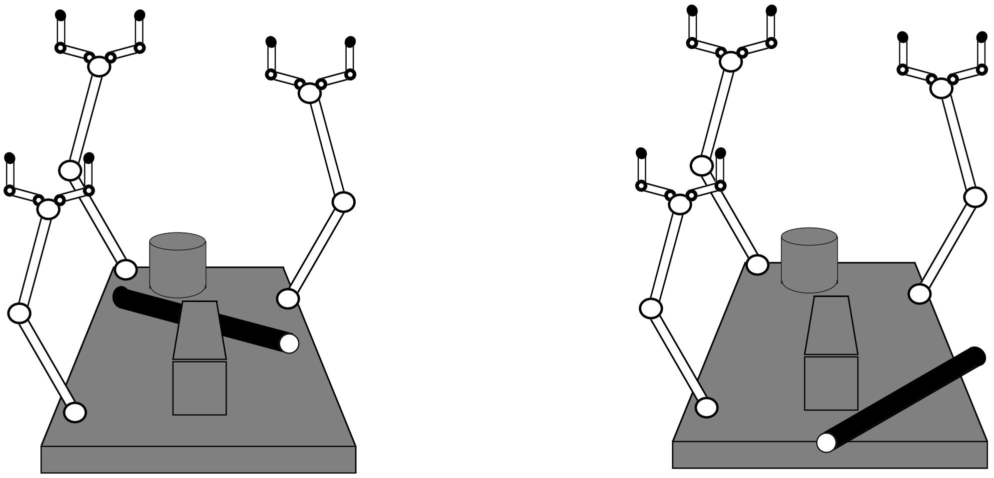
Consider the workcell shown in Fig. 2.1. It consists of a 3D workspace $\mathcal{W}$ with p robot arms $\mathcal{A}_i$ $(i = 1, \cdots, p)$, a single movable object $\mathcal{M}$, and q static obstacles $\mathcal{B}_j$ $(j = 1, \cdots, q)$. Let

- $\mathcal{C}_i$ be the C-spaces (configuration spaces) of the arms $\mathcal{A}_i$. Each $\mathcal{C}_i$ has dimension n_i, where n_i is the number of degrees of freedom of the robot $\mathcal{A}_i$.
- $\mathcal{C}_{obj}$ be the C-space of the object $\mathcal{M}$. $\mathcal{C}_{obj}$ is a 6-dimensional C-space.
- $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_p \times \mathcal{C}_{obj}$ be the composite configuration space of the whole system. Thus, a configuration in $\mathcal{C}$, called a **system configuration**, is of the form $(\boldsymbol{q}_1, \cdots, \boldsymbol{q}_p, \boldsymbol{q}_{obj})$, with $\boldsymbol{q}_i \in \mathcal{C}_i$ and $\boldsymbol{q}_{obj} \in \mathcal{C}_{obj}$.

We define the *C-obstacle region* $\mathcal{CB} \subset \mathcal{C}$ as the set of all system configurations where two or more bodies in $\{\mathcal{A}_1, \cdots, \mathcal{A}_p, \mathcal{M}, \mathcal{B}_1, \cdots, \mathcal{B}_q\}$ intersect.[1] We describe all bodies as closed subsets of $\mathcal{W}$; hence, $\mathcal{CB}$ is a closed subset of $\mathcal{C}$. The open subset $\mathcal{C} \setminus \mathcal{CB}$ is denoted by $\mathcal{C}_{free}$ and its closure by $cl(\mathcal{C}_{free})$.

The problem at hand is to manipulate the object $\mathcal{M}$ to some goal location. The possible system configurations where the object configuration is at its goal, lie on a cross section of $cl(\mathcal{C}_{free})$ defined by the goal object configuration. To make the

[1]We regard joint limits in $\mathcal{A}_i$ as obstacles that only interfere with the arms' motions.



(a) Initial System Config. **(b) Goal System Config.**

Figure 2.2: A manipulation task.

planning problem more specific we specifiy the goal location for the arms as well. The problem then becomes, "achieve some goal system configuration $\boldsymbol{q}^g_{sys}$, given an initial system configuration $\boldsymbol{q}^i_{sys}$" (see Fig. 2.2). The solution is a path within $cl(\mathcal{C}_{free})$ connecting these two configurations.

2.2 Manipulation Path

The term "path" has been used loosely up to here. Let us formally define this notion.

Definition 2.1 *A* **path** *is a* continuous map $\tau : [0, 1] \rightarrow \mathcal{C}$*, connecting two configuations $\tau(0)$ (the initial configuration) and $\tau(1)$ (the goal configuration).*

The path we seek has further restrictions. For example, since the object $\mathcal{M}$ requires some robots to grasp and manipulate it, a path τ that has the movable object flying on its own is not acceptable. Below we provide a formal definition of an acceptable path. We call it a **manipulation path**.

There are two constraints that must be satisfied in the manipulation problem. The first constraint is that the movable object can move only when the robots act on it and manipulate it accordingly. To be general, acting on an object may mean grasping it in a firm manner, or perhaps pushing it. The second constraint is that when the object is not moving, it is in a stable configuration.

For the most part we require that the arms, object, and obstacles not contact one another. However to satisfy the aforementioned constraints, we allow the end-effector of each arm manipulating $\mathcal{M}$ to make contact with it. Furthermore, $\mathcal{M}$ may also make contact with the environment during the manipulation phase (i.e., sliding on a table while a robot arm pushes it). For static stability, $\mathcal{M}$ may contact the end effector of stationary arms and obstacles. No other contacts are allowed. This leads us to define two subsets of $cl(\mathcal{C}_{free})$:

Definition 2.2 *The* **grasp space** $\mathcal{C}_{grasp}$ *is the set of all configurations in* $cl(\mathcal{C}_{free})$ *where one or several arms contact* $\mathcal{M}$ *in such a way that they have sufficient torque for quasi-static motion. The contact must be such that the object* $\mathcal{M}$ *does not slip away from the robots.*

By quasi-static motion, we mean a motion where the velocity and acceleration are small. In the case of robots firmly grasping $\mathcal{M}$, an example configuration in $\mathcal{C}_{grasp}$ would be one where the object does not slip out of the grasp of the end effectors, and the arms have sufficient torque to allow quasi-static motion in any kinematically feasible direction.

For a configuration to be in the grasp space the appropriate robots must execute a GRASP operation. This would be the necessary action to ensure that the object does not slip away from the robots (e.g., closing the fingers of the gripper around the object). GRASP is approximated here as an on/off switch. Notice that the configurations in $\mathcal{C}_{grasp}$ lie on constraint submanifolds in $\mathcal{C}$ defined by the rigid grasp relation between the robots and the movable object.

Definition 2.3 *The* **stable space** $\mathcal{C}_{stable}$ *is the set of all configurations in* $cl(\mathcal{C}_{free})$ *where* $\mathcal{M}$ *is statically stable.* $\mathcal{M}$*'s stability may be achieved by contacts between* $\mathcal{M}$ *and the arms and/or the obstacles.*

Notice that $\mathcal{C}_{grasp} \subseteq \mathcal{C}_{stable}$.

The manipulation path we seek must lie within these subsets of $cl(\mathcal{C}_{free})$. To clarify this notion we define transfer and transit paths.

Definition 2.4 *For a single fixed grasp relation, a* **transfer path** *is defined as the motion of the arms that moves $\mathcal{M}$.*

It lies in a cross-section of $\mathcal{C}_{grasp}$ defined by the attachment of $\mathcal{M}$ to the end effector of the manipulating arms. During a transfer path, not all moving arms need grasp $\mathcal{M}$; for example, some arms may be moving to allow the grasping arms to move without collision.

Definition 2.5 *A* **transit path** *is defined as the motion of the arms that does not move $\mathcal{M}$.*

Along such a path, $\mathcal{M}$'s static stability must be achieved by contacts with obstacles and/or stationary arms. Examples of such a path involve moving an arm to a configuration where it can grasp $\mathcal{M}$ or moving an arm to change its grasp of $\mathcal{M}$. A transit path lies in the cross-section of $\mathcal{C}_{stable}$ defined by the current fixed configuration of $\mathcal{M}$.

The manipulation path we seek consists of these transit and transfer paths that lie within $cl(\mathcal{C}_{free})$ (see Fig. 2.3).

Definition 2.6 *A* **manipulation path** *is a finite alternating sequence $(\tau_1, \tau_2, \ldots, \tau_{2p+1})$ for non-negative integer p, such that:*

- $\tau_1, \tau_3, \ldots, \tau_{2p+1}$ *are transit paths.*
- $\tau_2, \tau_4, \ldots, \tau_{2p}$ *are transfer paths.*
- *for every $l \in \{1, \ldots, 2p\}, \tau_l(1) = \tau_{l+1}(0)$.*

At the end of a transit path when it attaches to a transfer path, at least one robot is executing a GRASP operation. At the start of a transit path where robots detach

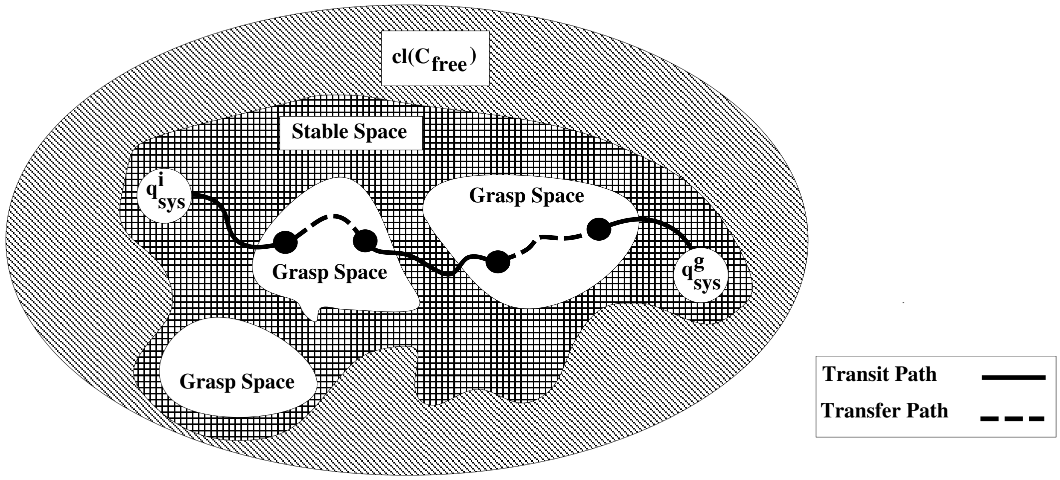


Figure 2.3: Components of a manipulation path and their relation to the subspaces of $cl(\mathcal{C}_{free})$.

from the object $\mathcal{M}$, then an UNGRASP operation is executed. UNGRASP is the action needed to undo a previous GRASP operation.

In a multi-arm manipulation planning problem, the geometry of the arms, movable object, and obstacles is given, along with the location of the obstacles. The goal is to compute a manipulation path between two specified system configurations, $\boldsymbol{q}^i_{sys}$ and $\boldsymbol{q}^g_{sys}$ respectively.

2.3 Example Manipulation Paths

Having defined what a manipulation path is, we now consider some example paths to clarify the terms introduced in the previous sections. We first look at a problem in a two-dimensional workspace as shown in Fig. 2.4. The movable object is a long bar AB (shown as a bold line) that can be moved in the plane by two identical arms, each with 3 revolute joints. The object can translate and rotate yielding a composite configuration space of dimension nine. The arms can grasp the object by positioning their endpoints at the extremities of the bar (then the grasp contacts behave as passive revolute joints) and they have sufficient torque to move the bar from any collision-free configuration. The grasp space $\mathcal{C}_{grasp}$ is then all collision-free configurations where

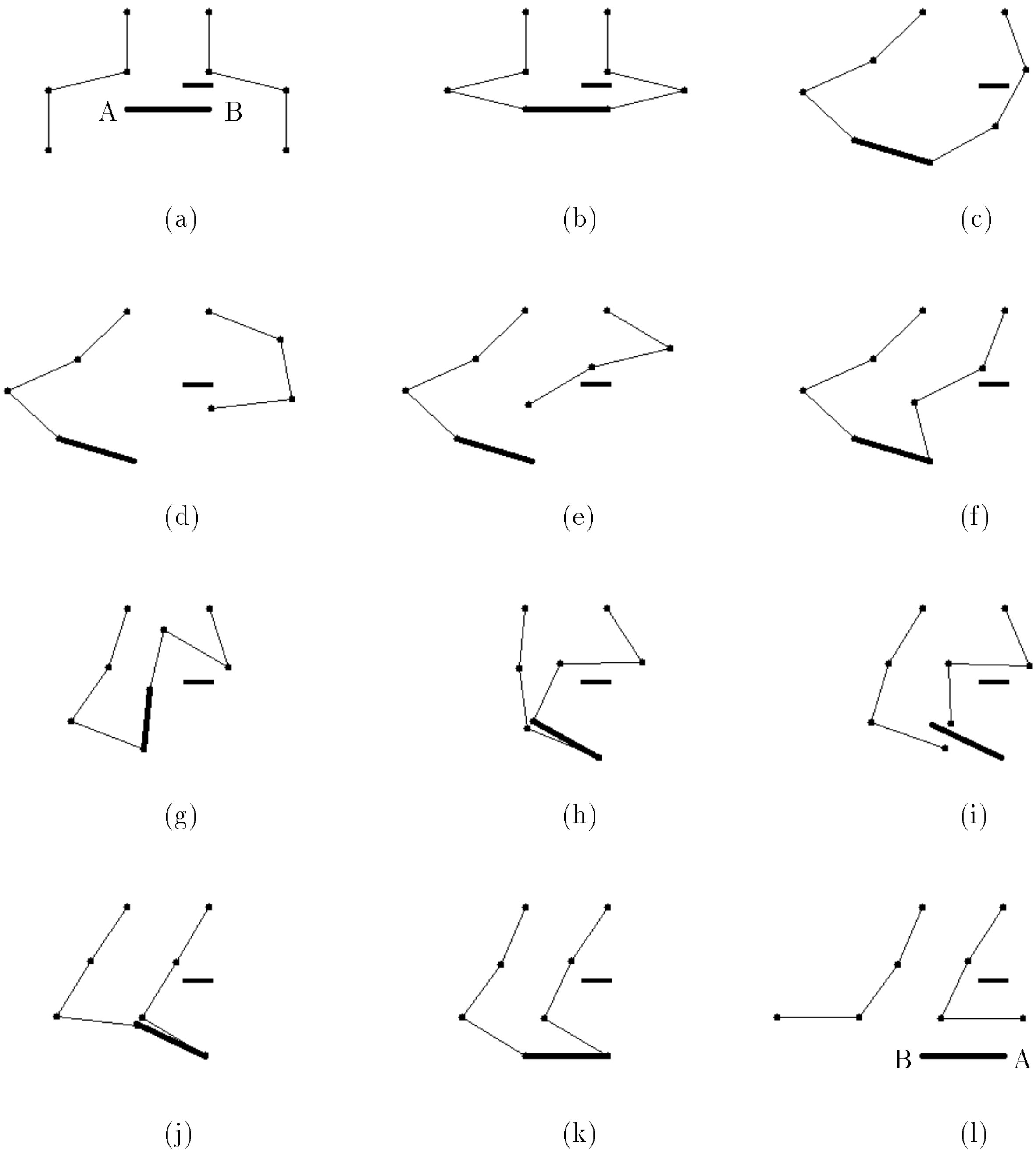


Figure 2.4: A manipulation path in a 2D workspace.

both arms are grasping the object. In this two-dimensional workspace we assume that the manipulated object slides on a table, hence every object configuration is stable and every configuration in $cl(\mathcal{C}_{free})$ is in the stable space $\mathcal{C}_{stable}$. The inital and goal system configuration, $\boldsymbol{q}^i_{sys}$ and $\boldsymbol{q}^g_{sys}$, are shown in snapshots (a) and (l) respectively, with one possible manipulation path to solve the task illustrated by snapshots (a) through (l). The two arms first execute a transit path to grasp the bar as shown in (b). A transfer path is then executed where the arms manipulate the object towards the goal, but a pending collision with the obstacle (c) yields the right arm to execute a transit path by ungrasping the bar (d), moving around the obstacle (e), and then regrasping the bar at a new configuration (f). The motion of the bar is resumed (g) (transfer path) until a collision between the two arms (h) requires them to swap their grasp positions (i) (transit path). The motion of the bar is resumed again (j) and the goal configuration of the bar is achieved (k) (transfer path). The two arms then ungrasp the bar and move to their final configurations (l) (transit path).

A manipulation problem in a three-dimensional workspace is more difficult since regrasping becomes more restrictive since not all object configurations are stable. Indeed, in a two-dimensional workspace the arms can let go of the object at any configuration and have the object stay, whereas in a three-dimensional workspace the object may fall down. We consider the problem of manipulating a L-shaped object in the three-dimensional workspace (Fig. 2.5). In this example, there a three robot arms each with six degrees of freedom and a L-shaped object, also with six degrees of freedom. The result is a composite configuration space of dimension twenty four. The object is heavy and requires two arms to actually carry and move it, but is still light enough that one arm can hold it in a static manner. The $\mathcal{C}_{grasp}$ is then all collision-free configurations where at least two arms are holding the object. $\mathcal{C}_{stable}$ is all configurations where the object is stable such as resting against the obstacles, but this also includes those configurations where one arm is holding the object in a collision-free manner. The inital and goal system configuration, $\boldsymbol{q}^i_{sys}$ and $\boldsymbol{q}^g_{sys}$, are shown in snapshots (a) and (l) respectively, with one possible manipulation path to solve the task illustrated by snapshots (a) through (l). Two arms first execute a transit path to grasp the bar as shown in (b). A transfer path is then executed where the arms manipulate the object towards the goal, but a pending collision with the obstacle (c) yields one arm to let go (d) and another arm to grasp the object (e) (transit path). The motion of the object is resumed where its threaded through the hole in the wall (transfer path) but again a pending collision causes the right arm to regrasp (f) (transit path). The object is brought through the hole (g) (transfer path) and a final regrasp is executed to the achieve the necessary grasp (h-k) to place the object at its goal location (transit path). Once the object is placed on the floor (transfer path) the arms let go and move to their final configuration (transit path). Notice that for every regrasping action at least one arm is holding the object in a stable manner.

The next two chapters focus on developing a practical approach to solving this multi-arm manipulation planning problem.

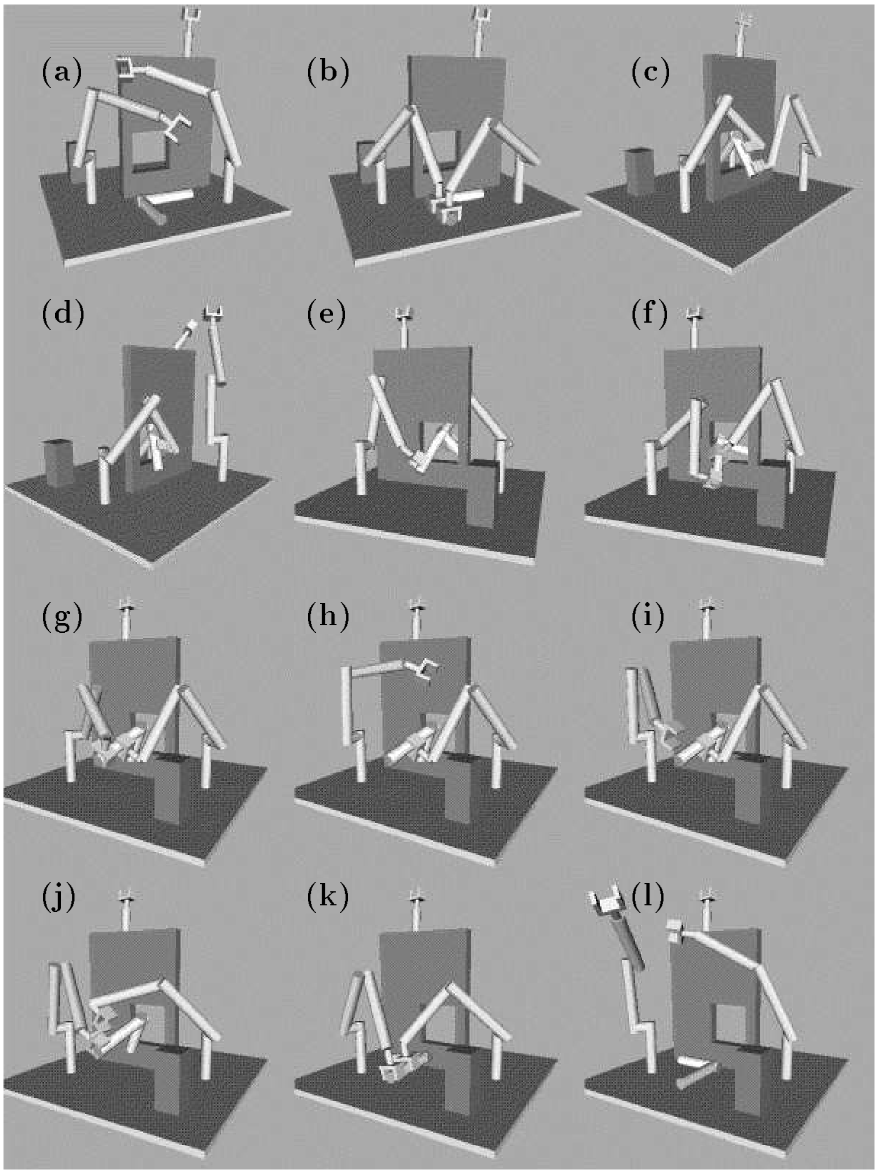


Figure 2.5: A manipulation path in a 3D workspace.

Chapter 3

Manipulation Planning in a 2D Workspace

In the previous chapter we formalize the multi-arm manipulation planning problem. Having defined what a manipulation path is, the next question is how do we find it? To actually compute a path, many questions need to be answered. For example, which arms should be responsible for manipulating the object, how should they grasp the object, and where should the arms ungrasp and then regrasp the object? One method to deal with all these questions in a unified framework is the construction and search of what is known as the *manipulation graph*. This is an idea originally proposed by Laumond and Alami [Laumond and Alami, 1988].

The basic idea is to first determine the connectivity of the free space by manipulation paths using Collins' decompositon algorithm, and then searching within the representation (which has a graph structure) for an actual manipulation path. Unfortunately due to the double-exponential time complexity of Collins' decomposition algorithm, this approach is impractical for dealing with robot systems with many degrees of freedom.

Our approach is a local one [Latombe, 1992] that consists of placing a fine resolution grid over the continuous configuration space, and then utilizing a heuristically-guided local search over the grid to obtain the solution. Our motivation for taking this approach comes from the great success its had in dealing with the piano movers' problem (the basic path planning problem of moving the robot from one configuration to another). In particular there exist two efficient planners called the *best-first planner* (BFP) and the *randomized path planner* (RPP) [Barraquand and Latombe, 1991b]. BFP is a fast, resolution-complete planner well suited to solving the piano movers' problem for robots with three degrees of freedom or less. RPP is a practical path planner that for many complicated examples finds the collision-free paths of robots with many degrees of freedom. We refer the reader to Chapter 7 of [Latombe, 1991] for details of the BFP and RPP algorithm.

In this chapter, we make an *in-depth* study of how the local approach to motion planning can be applied to a simplified multi-arm manipulation planning problem. The specific example is for two robot arms and a single movable object in a two-dimensional workspace. This scenario comes from the dual-arm robot system developed in the Aerospace Robotics Laboratory, Stanford University [Pardo et al., 1993]. We develop techniques for treating this complex manipulation problem (as defined in Chapter 2) as a form of the more basic piano movers' problem, and then solve it using BFP. The resulting implemented planner is resolution-complete. The exciting result of this study is that it gives us a framework for developing a practical manipulation planner. Indeed, using the same techniques, RPP can be used to solve many complex instances of the transformed problem of manipulation planning in a three-dimensional workspace.

3.1 The Scenario

We consider the following scenario (this is different from the specific 2D example used in Chapter 2):

- Two SCARA-type robot arms work in an environment where the obstacles are cylindrical bodies of infinite height.

- Each arm has four degrees of freedom; shoulder, elbow and wrist revolute joints, and a prismatic joint to move the end-effector up and down. In addition, from the top view of the robots, the end-effector and the prismatic joint are occluded

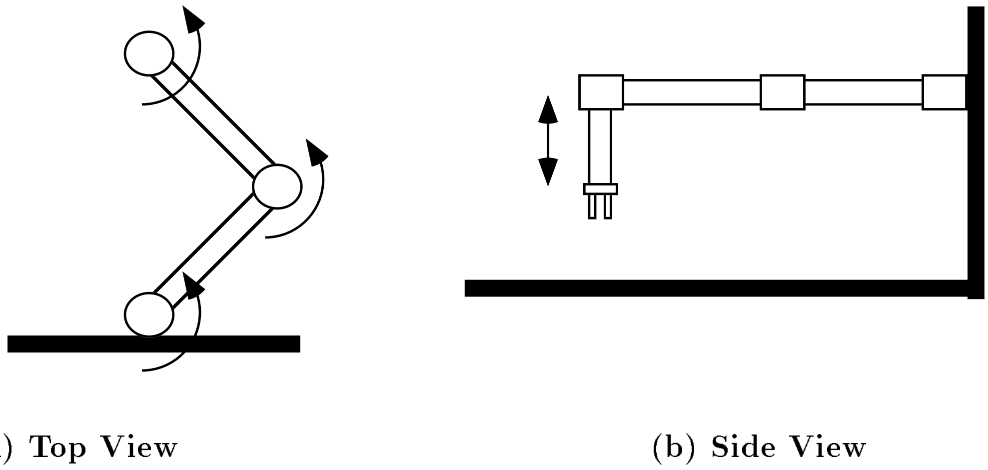


Figure 3.1: The robots for the planar workspace.

by the links of the forearm (see Fig. 3.1).

- The robots grasp the movable object $\mathcal{M}$ from above. Thus, the links of the arms never collide with $\mathcal{M}$.

- *Both* arms must firmly grasp $\mathcal{M}$ for it to move. Thus, the arms form a closed-kinematic-chain when they manipulate it.

- Any closed-chain configuration where both arms are grasping $\mathcal{M}$ is an element of $\mathcal{C}_{grasp}$.

- All object configurations are statically stable regardless of whether the arms are grasping it.

Notice that because of the structure of each arm, whenever the upper-arm and fore-arm are collision-free, the prismatic link and end-effector are also collision-free. Thus, from the planning viewpoint, the up/down, and rotational degrees of freedom of the end effector of both arms can be ignored. Consequently, the manipulation planning problem is for two robots, each with two degrees of freedom and a two-dimensional workspace. The object has three degrees of freedom (two translational and one rotational degree of freedom) resulting in a seven dimensional composite

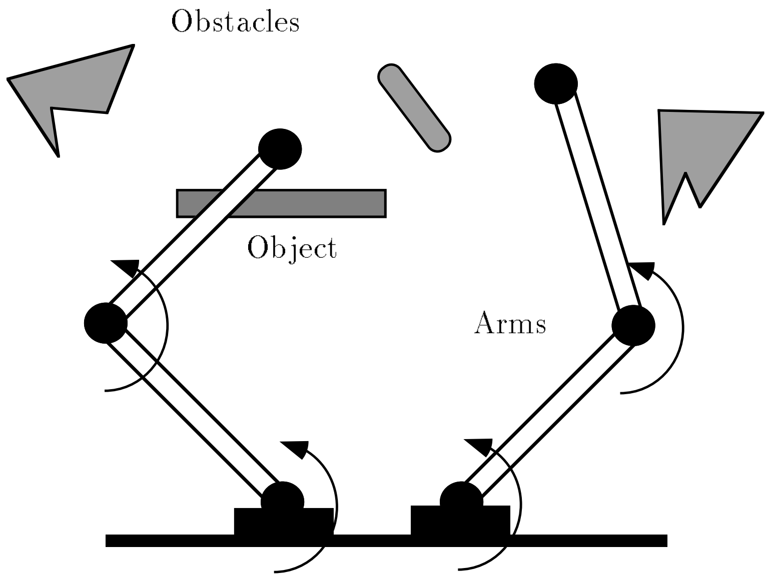


Figure 3.2: The robots for the planar workspace.

configuration space. The motion of the robots and the movable object are constrained by the obstacles. However, acting alone the robots are *not* constrained by the movable object since they move in a plane above the object. This is illustrated in Fig. 3.2.

In [Koga and Latombe, 1992] we describe a method to explicitly build the manipulation graph to solve this two-dimensional problem. The method is an implementation of the ideas proposed in [Alami et al., 1990] and uses a discretized approach rather than Collins' decomposition algorithm. In particular, the manipulation graph is created by first determining the connectivity of the discretized freespace by transfer paths and then linking these connected components by transit paths. The approach is well suited to this two-dimensional problem but due to memory size limits it is unclear how to extend it to the more complex problem of dealing with the three-dimensional workspace. We instead develop a new method, keeping in mind that eventually we need to deal with the complex three-dimensional case.

3.2 Overview of the Approach

As well demonstrated by BFP and RPP, the local approach to motion planning is well suited to solving the piano movers' problem. With this in mind, we opt to solve the above problem in three steps:

1. Find a collision-free path for the object (call it τ_{obj}).
2. Compute a sequence of transfer paths where the object tracks τ_{obj}.
3. Connect the transfer paths with transit paths to complete the manipulation path.

Steps (1) and (3) are instances of the piano movers' problem.

Although intuitively simple, there are complications that need to be addressed. Specifically, we must impose constraints on step (1) to guarantee the completion of steps (2) and (3). For example, an object path that has the object move out of the robot workspace would be an unacceptable outcome of step (1).

How do we make such a guarantee? To guarantee the completion of step (2), every object configuration along the path τ_{obj} requires that both arms are able to grasp it in a collision-free manner. To further guarantee the completion of step (3), we need the transfer paths from step (2) to be *connectable.*

Definition 3.1 *Two transfer paths τ_a and τ_b are said to be* **connectable** *if there exists $\boldsymbol{q}^a \in \tau_a$ and $\boldsymbol{q}^b \in \tau_b$ with the same object configuration and with arm configurations lying in the same connected component of $\mathcal{CF}_{arms}$ ($\mathcal{CF}_{arms}$ is the freespace of the composite C-space of both arms).*

Thus, if two transfer paths are connectable, then a transit path linking them together is guaranteed to exist.

There is one last constraint that needs to be satisfied. The first and last transfer path extracted from τ_{obj} (call them τ_{first} and τ_{last} respectively) must be such that the arm configurations at $\tau_{first}(0)$ and $\tau_{last}(1)$ lie in the same connected component of $\mathcal{CF}_{arms}$ as the arm configurations of $\boldsymbol{q}^i_{sys}$ and $\boldsymbol{q}^g_{sys}$, respectively. This ensures the existence of a transit path to connect the inital and goal system configurations to τ_{first} and τ_{last}, respectively.

The rest of this chapter dicusses the details of our algorithm to solve the above problem using these three steps. Many of these ideas will be extended in Chapter 4 to develop a practical manipulation planner in three-dimensional workspaces.

3.3 Dealing with the Constraints

To allow considerations of the added constraints that step (1) must satisfy, two tools are constructed. These tools further impose a discretization of the search space. Our method is to use these tools in conjunction with BFP to find the special τ_{obj}.

Definition 3.2 *A* **grasp assignment** *is a specification of the point on the object that each arm must grasp and the type of the inverse kinematics solution for the respective arms (i.e., elbow in or elbow out).*

Therefore, given an object configuration and a grasp assignment, the configuration of the arms grasping the object is specified. Notice that for each robot arm there are two solutions to the inverse kinematics problem.

Tool 1: One constraint identified in the previous section is that the arms must grasp the object at all points of the object path. There is an infinite number of ways to grasp the object, with each grasp defining a constraint submanifold in $\mathcal{C}$. Since the objective is to find an object path from which the sequence of transfer paths are extracted, then extracting an infinite number of them is potentially computationally explosive. With this is mind, we restrict the number of ways in which the arms can grasp the object.

Definition 3.3 *The* **grasp set** $\mathcal{G}$ *is the finite set of grasp assignments from which all possible ways for the arms to grasp the object can be selected.*

This set can be predefined by the user, or automatically computed using some grasp planner (there are a variety of existing algorithms that could be used [Pertin-Troccaz, 1989]). Thus, for every configuration in τ_{obj} there must be at least one grasp assignment in $\mathcal{G}$ which yields a collision-free system configuration. This is a necessary but not sufficient condition.

Tool 2: To determine whether two transfer paths are *connectable*, the planner must have the ability to determine whether two collision-free configurations of the arms lie in the same connected component of $\mathcal{CF}_{arms}$. This desired information comes from the topology of the composite C-space of both arms (call it $\mathcal{C}_{arms}$). To simplify matters, the freespace is approximated by placing a fine regular grid over $\mathcal{C}_{arms}$ and then exhaustively pre-computing whether each discrete configuration is collision-free or not. Within this grid, we say that two configurations are adjacent if they are collision-free and differ by one discrete unit. By assuming that adjacent configurations can be linked by a straight line collision-free path in $\mathcal{C}_{arms}$ (a reasonable assumption with a fine enough discretization), the connected components of $\mathcal{CF}_{arms}$ are then the equivalence classes for the transitive closure of this adjacency relation. Because this information is in a bitmap format, we can take any two collision-free arm configurations and do a lookup to see whether they lie in the same connected component of freespace. The amount of storage required for this bitmap representation grows exponentially with the dimension of $\mathcal{C}_{arms}$.

Given these two tools, the problem of finding the desired object path τ_{obj} can now be addressed.

3.4 Finding the Object Path

We first describe the basic idea of BFP as it applies to finding the object path. To distinguish it from the general BFP we denote the algorithm as BFP*. This presentation is derived from Chapter 7 of [Latombe, 1991].

The three-dimensional configuration space $\mathcal{C}_{obj}$ is discretized into a fine regular grid and the path is found using a heuristically guided local search. The generalized coordinates used to describe the configuration of the object are x, y, and θ, where x and y are the translational terms and θ is the rotational term. We denote the grid as $\mathcal{GC}$. The heuristic is a potential field defined over $\mathcal{GC}$ having a global minimum at the goal object configuration $\boldsymbol{q}^g_{obj}$ (see Appendix A for a definition of this potential). The path of $\mathcal{M}$ is constructed from its initial object configuration $\boldsymbol{q}^i_{obj}$ by exploring collision-free neighboring configurations of $\mathcal{GC}$ in a best-first fashion. Given a configuration $\boldsymbol{q}_{obj}$ in $\mathcal{GC}$, its neighbour is defined as any configuration in $\mathcal{GC}$ having at most one coordinate differing from those of $\boldsymbol{q}_{obj}$ by one increment of the discretization. The potential field is used as the cost function and essentially guides the search to the global minima at $\boldsymbol{q}^g_{obj}$. To simplify the presentation we assume:

- Both $\boldsymbol{q}^i_{obj}$ and $\boldsymbol{q}^g_{obj}$ are in $\mathcal{GC}$.
- If two neighbors in $\mathcal{GC}$ lie in the freespace, then the straight line path in $\mathcal{C}_{obj}$ that connects them is also in the freespace.
- $\mathcal{GC}$ is bounded and forms a parallelepiped, but with the $\theta = 0$ and the $\theta = 2\pi$ faces being identical.

BFP* constructs the path by iteratively constructing a tree $\mathcal{T}$ whose nodes are configurations in $\mathcal{GC}$. With $\boldsymbol{q}^i_{obj}$ as the root of $\mathcal{T}$, at each iteration, BFP* identifies the leaf of $\mathcal{T}$ with the lowest potential (call it l_{best}) and then visits its neighbors not already in $\mathcal{T}$. If a neighbor is collision-free, then its potential value is determined and inserted into $\mathcal{T}$ as the successor to l_{best}. Each node in $\mathcal{T}$ has a pointer towards its parent. The algorithm terminates with success when $\boldsymbol{q}^g_{obj}$ is attained. Otherwise, it terminates with failure when the freespace of $\mathcal{GC}$ reachable from $\boldsymbol{q}^i_{obj}$ is fully explored. If the search is successful, the path τ_{obj} is generated by tracing the pointers in $\mathcal{T}$ from $\boldsymbol{q}^g_{obj}$ to $\boldsymbol{q}^i_{obj}$.

To ensure that the same configuration is not considered repeatedly, each visited configuration is marked as such, and never considered as a neighbor again. Hence, for any local minima in $\mathcal{GC}$, BFP* escapes the potential well by filling it up (which is reasonable in a three-dimensional grid).

We now describe how the other constraints on τ_{obj} are enforced.

Let $R(g, \boldsymbol{q}_{obj})$ be the resulting system configuration when an object configuration is paired with a grasp assignment g.

Definition 3.4 *The* **feasible grasp set** $\mathcal{FS}$ *for any given object configuration* $\boldsymbol{q}_{obj}$ *is the subset of* $\mathcal{G}$ *where* $\forall g \in \mathcal{FS}$, $R(g, \boldsymbol{q}_{obj})$ *is collision-free.*

For our simplified problem every arm configuration along the manipulation path must necessarily belong to the same connected component of $\mathcal{CF}_{arms}$, since the object $\mathcal{M}$ has no influence on the connectivity of $\mathcal{CF}_{arms}$. Thus, before starting the search a few tests are conducted. First, it is verified that the arm configurations for $\boldsymbol{q}^i_{sys}$ and $\boldsymbol{q}^g_{sys}$ lie in the same connected component of $\mathcal{CF}_{arms}$. If they do not, then a manipulation path cannot be found. If they do, then the connected component is labelled as $\mathcal{CC}_{arms}$. The second test involves the feasible grasp sets for the initial and goal object locations. We denote them $\mathcal{FS}^i$ and $\mathcal{FS}^g$, respectively. $\mathcal{FS}^i$ must have at least one grasp assignment g such that $R(g, \boldsymbol{q}^i_{obj})$ lies in $\mathcal{CC}_{arms}$. Similarly, $\mathcal{FS}^g$ must have at least one grasp assignment g such that $R(g, \boldsymbol{q}^g_{obj})$ lies in $\mathcal{CC}_{arms}$. If they do not, then there is no manipulation path. If they do, then $\mathcal{FS}^i$ is updated by keeping only those grasp assignments which yield configurations in $\mathcal{CC}_{arms}$.

After passing the tests, τ_{obj} can be computed using BFP*. For the configuration $\boldsymbol{q}_{obj}$ of l_{best} (initially, $\boldsymbol{q}^i_{obj}$), the algorithm considers every neighbor $\boldsymbol{q}'_{obj}$ of $\boldsymbol{q}_{obj}$ that have yet to be explored. For every $\boldsymbol{q}'_{obj}$, our version of BFP* checks that it is collision-free and if so, then computes its feasible grasp set and verifies that at least one element also lies in the feasible grasp set of $\boldsymbol{q}_{obj}$ (initially, for $\boldsymbol{q}^i_{obj}$ the associated feasible grasp set is $\mathcal{FS}^i$). Furthermore, it filters out any grasp assignment in the feasible grasp set of $\boldsymbol{q}'_{obj}$ which are not in $\mathcal{CC}_{arms}$. This ensures the existence of the transfer paths, with the transit paths to link them. If the resulting grasp set for $\boldsymbol{q}'_{obj}$ is non-empty, then $\boldsymbol{q}'_{obj}$ and its associated feasible grasp set are inserted into $\mathcal{T}$ as a successor of $\boldsymbol{q}_{obj}$. The search continues until $\boldsymbol{q}'_{obj} = \boldsymbol{q}^g_{obj}$ (success) or until all possible configurations are visited without reaching $\boldsymbol{q}^g_{obj}$ (failure).

If the algorithm succeeds, it returns a τ_{obj} satisfying the required constraints. The path τ_{obj} is described as a sequence of grid configurations. It remains to extract the series of transfer paths and then compute the interconnecting transit paths. We refer the reader to Appendix A for a formal expression of the BFP* algorithm.

3.5 Extracting the Manipulation Path

Embedded within τ_{obj} is the manipulation path. We first extract the sequence of connectable transfer paths.

The first configuration $\boldsymbol{q}^i_{obj}$ in τ_{obj} and its feasible grasp set is considered. Depending on the size of the feasible grasp set, there will be one or several grasp assignments associated with $\boldsymbol{q}^i_{obj}$. For each one, the maximal subpath of τ_{obj} is computed, starting at $\boldsymbol{q}^i_{obj}$ such that the same grasp assignment is associated with every configuration in this subpath. The planner selects the grasp configurations for $\boldsymbol{q}^i_{obj}$ that results in this longest subpath (call it τ^1_{obj} and let g_1 be the grasp assignment). τ^1_{obj} is transformed into a transfer path by taking the resulting configurations of the pairing of g_1 and the configurations in τ^1_{obj}. This is the first transfer path in the sequence.

The second transfer path is obtained as follows (assume that the end of τ_{obj} is not yet reached). Let $\boldsymbol{q}^c_{obj}$ be the last configuration of τ^1_{obj}. The planner selects the longest subpath τ^2_{obj} of τ_{obj} beginning at $\boldsymbol{q}^c_{obj}$, whose grasp assignment (call it g_2) remains valid throughout. τ^2_{obj} is transformed into a transfer path by taking the resulting configurations of the pairing of g_2 and the configurations in τ^2_{obj}. This is the second transfer path in the sequence. The search proceeds in the same way until the last configuration ($\boldsymbol{q}^g_{obj}$) of τ_{obj} is attained.

We now show that this greedy algorithm is guaranteed to extract a sequence of connectable transfer paths from τ_{obj}, and terminate at $\boldsymbol{q}^g_{obj}$.

Lemma 3.1 *The second transfer path is composed of at least two discrete object configurations.*

Proof: This is by construction of τ_{obj}. The feasible grasp set for the successor of $\boldsymbol{q}^c_{obj}$ in τ_{obj} has at least one grasp assignment in the feasible grasp set of $\boldsymbol{q}^c_{obj}$. Thus, in the worst case this common grasp assignment will be g_2, and the second transfer path will have two object configurations. $\square$

Corollary: *All the extracted transfer paths have at least two object configurations.*

Lemma 3.2 *For the given τ_{obj} the search for transfer paths will terminate at $\boldsymbol{q}^g_{obj}$.*

Proof: By the corollary to Lemma 3.1, in the worst case each transfer path will have only two configurations. Neighboring paths share only one configuration, thus the search can proceed until the last configuration in τ_{obj} is reached. □

Lemma 3.3 *The first and second transfer paths are connectable.*

Proof: The grasp assignments g_1 and g_2 both belong to the feasible grasp set of $\boldsymbol{q}^c_{obj}$ and the resulting arm configurations lie in $\mathcal{CC}_{arms}$. □

Corollary: *The sequence of extracted transfer paths are all connectable.*

Having extracted the transfer paths $(\tau_2, \tau_4, \ldots, \tau_{2p})$, the next step is to find the transit paths $(\tau_1, \tau_3, \ldots, \tau_{2p+1})$ to complete the manipulation path. The first transit path (τ_1) connects $\boldsymbol{q}^i_{sys}$ to the configuration $\tau_2[0]$ (the arm motion to grab the object at its initial configuration). The last transit path (τ_{2p+1}) connects the configuration $\tau_{2p}[1]$ to $\boldsymbol{q}^g_{sys}$ (the arm motion to let go of the object at its goal location). The remaining transit paths τ_{2i+1} connects the configuration $\tau_{2i}[1]$ to $\tau_{2i+2}[0]$ $(i = 1, \ldots, p-1)$. The resulting sequence, $\tau_1, \tau_2, \ldots, \tau_{2p+1}$, is the manipulation path connecting $\boldsymbol{q}^i_{sys}$ to $\boldsymbol{q}^g_{sys}$.

Finding these transit paths is the simple matter of searching $\mathcal{CC}_{arms}$. In our implementation, for each discrete configuration in $\mathcal{CC}_{arms}$ we previously encode a path that connects it to a *seed* configuration in $\mathcal{CC}_{arms}$ $(\boldsymbol{q}^{seed})$. Consequently, to find a path between any two configurations $\boldsymbol{q}^a$ and $\boldsymbol{q}^b$, it is simply a matter of concatenating the two paths between $\boldsymbol{q}^a$ and $\boldsymbol{q}^{seed}$ and between $\boldsymbol{q}^{seed}$ and $\boldsymbol{q}^b$, and then smoothing the path to yield a more direct motion (see Appendix B for the smoothing algorithm). The encoding is achieved during the precomputation phase of identifying the connected components of $\mathcal{CF}_{arms}$. We refer the reader to Appendix B for a thorough explanation of how the connected components are identified and how the path data is encoded to it members.

Proposition 3.1 *From τ_{obj} we can extract a manipulation path connecting $\boldsymbol{q}^i_{sys}$ to $\boldsymbol{q}^g_{sys}$.*

Proof: By Lemma 3.2 and the corollary to Lemma 3.3, a sequence of connectable transfer paths can be extracted from τ_{obj}. Furthermore, the arm configurations at

$\boldsymbol{q}^i_{sys}$, $\boldsymbol{q}^g_{sys}$ and at all configurations of the transfer paths lie in $\mathcal{CC}_{arms}$. Thus, the necessary arm motions (transit paths) can be constructed. □

It may occur that a subpath τ^i_{obj} has more that one grasp assignment g_i associated with it. It does not matter which one is used since the resulting arm configurations will all lie in $\mathcal{CC}_{arms}$. However, there may be some criteria that makes one manipulation path better than another. In this case, all the possible manipulation paths can be constructed, and then evaluated to choose the most appropriate one.

3.6 Details

There is one potential problem in the above approach. When either arm is close to the singular position where the two links of the arm are almost superimposed, then any small change in the gripper location may cause a dramatic change in the joint angles (the links may sweep a large area). Hence, for such a situation, when searching for the object path, though the pairing of the two consecutive object configurations and the grasp assignment yields system configurations that are collision-free, there may be no collision-free path of the closed-loop chain between them. We illustrate this failure in Fig. 3.3. For the start and end of a small displacement of the object, the system takes the collision-free configurations shown in Fig. 3.3 (a) and (b). However, no collision-free path exists to actually link them together. We deal with this problem as follows. We say that two system configurations are close to each other if and only if the change in the joint angles of the arms is within some threshold (and of course they have the same grasp assignment). During the search for the path τ_{obj}, when a configuration $\boldsymbol{q}'_{obj}$ is considered as a potential successor of $\boldsymbol{q}_{obj}$ (see above), it is verified that for each common grasp assignment g in the feasible grasp sets of $\boldsymbol{q}'_{obj}$ and $\boldsymbol{q}_{obj}$, that $R(g, \boldsymbol{q}'_{obj})$ and $R(g, \boldsymbol{q}_{obj})$ are *close to each other*. If they are not close to each other, an additional test is conducted to verify that the arm configurations are collision-free between $\boldsymbol{q}_{obj}$ and $\boldsymbol{q}'_{obj}$. This is done by interpolating between these two object configuration with a fine enough discretization that the successive arm configuration resulting from the pairing of g and the discretized object configurations are close to each other. If any

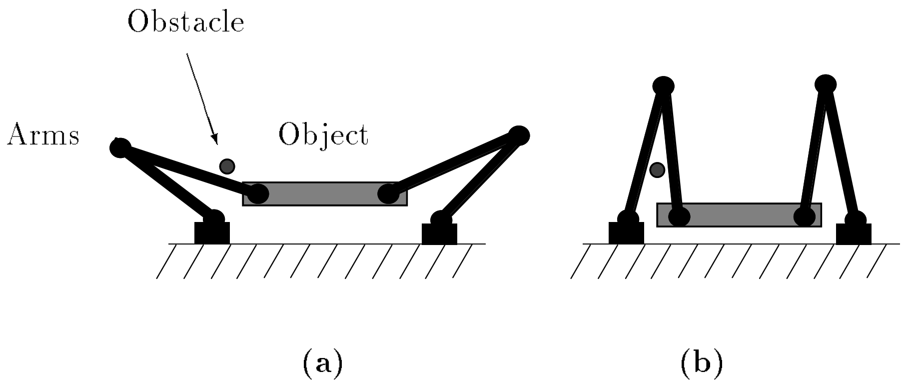


Figure 3.3: Restriction on the object path.

arm configuration is in collision then this grasp assignment for $\boldsymbol{q}'_{obj}$ is deleted from its feasible grasp set.

3.7 Minimizing the Number of Regrasps

The algorithm we have presented will find a manipulation path, but with no regard to minimizing the number of regrasp operations. Since regrasping can be a time consuming operation for real robots, it might be desirable to find manipulation paths that minimize the number of regrasps.

One method to minimize the number of regrasps would be to construct the manipulation graph and then find the path that minimizes the number of regrasping motions. We present a local approach to solving this problem; it is a simple extension of the previous algorithm. We simply augment the potential field with an appropriate penalty function for the number of regrasps and with a slight further modification to BFP*, we ensure that τ_{obj} is such that the extracted manipulation path minimizes the number of regrasps.

The slight modification is as follows. Each configuration $\boldsymbol{q}_{obj}$ in $\mathcal{GC}$ has six neighbors (recall, we defined a neighbour of $\boldsymbol{q}_{obj}$ as any configuration in $\mathcal{GC}$ having one coordinate differing from those of $\boldsymbol{q}_{obj}$ by one increment of the discretization). We modify BFP* to mark a configuration as visited such that a configuration can not be visited from the same parent configuration twice. This means that a best-first search can insert the same configuration in $\mathcal{T}$ a multiple of times, but never with the same parent configuration. Furthermore, for each object configuration visited in the search we keep a record of certain grasp assignments from its feasible grasp set and call it the **special feasible grasp set** $\mathcal{FS}^\star$. For the initial object configuration, its $\mathcal{FS}^\star$ is in fact just the feasible grasp set. For the other configurations, its $\mathcal{FS}^\star$ is the intersection of its feasible grasp set $\mathcal{FS}$ and the $\mathcal{FS}^\star$ of its parent configuration. By keeping a $\mathcal{FS}^\star$ for each visited configuration, whenever it becomes empty, we know that a regrasp action is required at its parent configuration. In this case, the $\mathcal{FS}^\star$ for this configuration is reset to equal the intersection of its $\mathcal{FS}$ and its parent configurations $\mathcal{FS}$. We thus have a method to determine the number of regrasping steps required to get to any configuration along a specific path. This is essentially a sequential version of the greedy algorithm to extract the transfer paths from τ_{obj}. Finally, we modify BFP* so that the search terminates with success when the $\boldsymbol{q}_{obj}$ of l_{best} matches $\boldsymbol{q}^g_{obj}$ (originally the search terminates with success when $\boldsymbol{q}'_{obj} = \boldsymbol{q}^g_{obj}$). This may cause some extra configurations to be considered but it guarantees that the resulting path has the minimum number of grasps.

Proposition 3.2 *Let N be the maximum value for the potential field on $\mathcal{GC}$. Let the potential value assigned to each node in $\mathcal{T}$ be the sum of the value from the original potential field plus a penalty equal to N times the number of regrasps required to get from the current node to the root node. Using the modifed BFP* it is guaranteed that the successful termination of the search results in a path with the minimum number of regrasps.*

Proof: This is due to the penalty function added to the potential. From the potential value associated to each node in $\mathcal{T}$ we can determine how many regrasp operations there will be in tracing back to $\boldsymbol{q}^i_{obj}$. If the potential value is between p and $(p{+}1)N{-}1$ then there will be p regrasp operations. Consequently, given a choice between a leaf node with p regrasps or $p+1$ regrasps for l_{best}, BFP* will always choose the node with p regrasps since its associated potential value will always be lower. Therefore, all possible configurations with p regrasps will be considered before any with $p+1$ regrasps. Since BFP* starts the search at $\boldsymbol{q}^i_{obj}$, each new configuration that appears as l_{best}, will be such that it traces back to $\boldsymbol{q}^i_{obj}$ in the minimum number of regrasp operations. Hence, when the the configuration for l_{best} is the goal configuration (the search is terminated) the resulting path has the minimum number of regrasp actions. $\square$

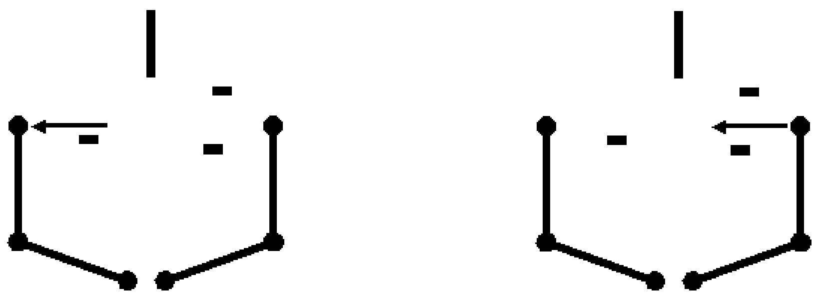
3.8 Resolution Completeness

Proposition 3.3 *The 2D Manipulation Planner is resolution complete, that is it will find a manipulation path if it exists and will report failure otherwise (for the given discretization).*

Proof: If no τ_{obj} exists between $\boldsymbol{q}^i_{obj}$ and $\boldsymbol{q}^g_{obj}$, then the modified BFP* will visit every possible reachable object configuration from $\boldsymbol{q}^i_{obj}$ and then terminate (reports failure). Clearly, if no τ_{obj} exists then a manipulation path cannot exists. Since BFP* in the worst case will visit all possible reachable configurations from $\boldsymbol{q}^i_{obj}$, then if a τ_{obj} exists for the given $\mathcal{GC}$, BFP* will find it. By proposition 3.1, a manipulation path will be extracted. $\square$

Proposition 3.4 *The worst-case time complexity of the 2D Manipulation planner is $O(d^4 + sd^3)$, where d is the number of discretization points for each axis of $\mathcal{C}$ and s is the number of grasp assignments in the grasp set $\mathcal{G}$.*

Proof: The discretized $\mathcal{C}_{arms}$ has size $O(d^4)$. The time complexity to construct $\mathcal{CF}_{arms}$ is then $O(d^4)$ since it is a systematic search of the discretized $\mathcal{C}_{arms}$. To determine the connected components and their encoding to generate a collision-free path between any two configuration within it, is also $O(d^4)$ (see Appendix B). We can represent $\mathcal{T}$ as a balanced tree [Aho, Hopcroft, and Ullman, 1983] so that inserting each configuration in $\mathcal{T}$ and extracting l_{best} takes logarithmic time in the number of elements in $\mathcal{T}$. In addition, it takes in the worst-case s operations (the feasible grasp set $\mathcal{FS}$



(a) Initial Configuration (b) Goal Configuration

Figure 3.4: The manipulation task.

is $\mathcal{G}$) to determine whether a configuration should be added into $\mathcal{T}$. The worst-case size of $\mathcal{T}$ is $O(d^3)$, hence the time complexity of the procedure BFP* as described above is $O(d^3 \log d + sd^3)$. Furthermore, to find the sequence of transfer paths and to determine the corresponding transit paths is both $O(q)$, where q is the number of discrete points making up each path. Thus, the time complexity of the 2D manipulation planner is $O(d^4 + sd^3)$. $\square$

3.9 Examples

We show the output of our implemented planner for the case where the number of regrasps are not minimized, and for the case where they are. The same movable object and environment, and the task of moving the object from the left side of the workspace to the right side are considered for both cases (see Fig. 3.4).

The grasp set $\mathcal{G}$ consists of 24 grasp assignments, resulting from the possible combination of arms, inverse kinematics solutions and the three locations on the object for grasping; the front, middle, and end. The planner is implemented in the C programming language and runs on a DEC Alpha workstation under UNIX.

Fig. 3.5 shows the path found without minimizing the number of regrasps. This path was found using a workspace discretization of 200×200 for the translation degrees of freedom of the object. The rotation angle of the object is discretized into 128 pieces. Each arm joint angle is discretized into 0.0491 radian pieces yielding a 4D arm C-space grid of $128 \times 128 \times 128 \times 128$ points. The joint limits for the first link of the arms are when they are horizontal, and the joint limits for the second link are when they are within 0.15 radians of the first link. Notice that seven regrasp operations are required. Computing the arm freespace took on the order of ten minutes, while the search for the path took around 5 seconds.

Fig. 3.6 shows the path found while minimizing the number of regrasp operations. The environment, discretization, joint limits, and the inital and goal locations are the same as above. In this case only two regrasps were required. The path was found in around 10 seconds after the preprocessing stage.

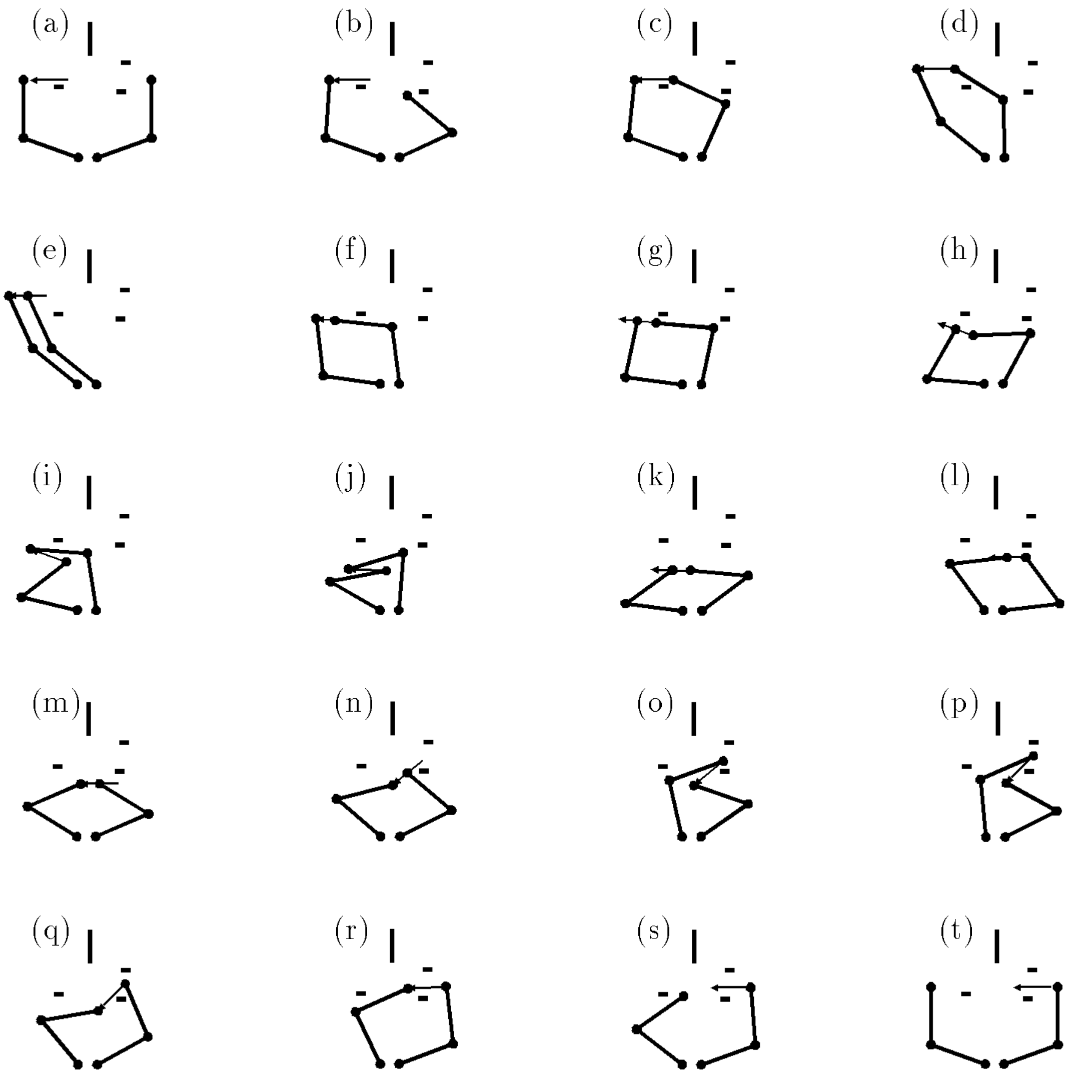


Figure 3.5: A manipulation path.

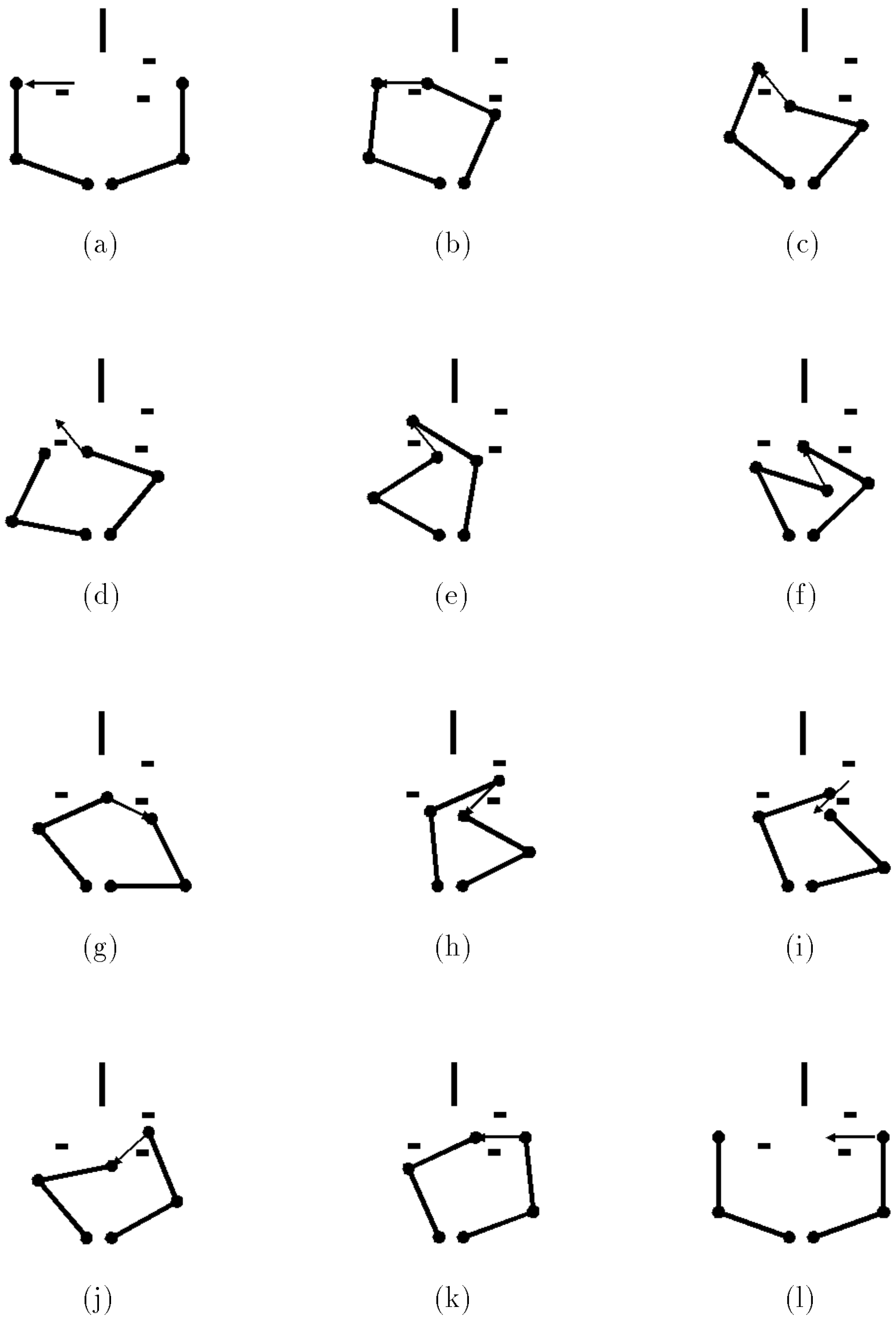


Figure 3.6: A manipulation path with minimal regrasping.

Chapter 4

Manipulation Planning in a 3D Workspace

In this chapter we extend the methods for solving the two-dimensional problem to the more challenging and interesting case of multi-arm manipulation planning in a three-dimensional workspace; the three-dimensional case is where most realistic manipulation tasks occur. The environment consists of two or more robot arms, a movable object, and fixed obstacles. The result is a practical planner that reliably finds manipulation paths for a large variety of realistic problems. We demonstrate the effectiveness of the algorithm by applying the implemented planner to different examples.

4.1 Extending the Ideas from the 2D Case

We use the three step approach outlined in Chapter 3 as the basis for a practical manipulation planner. That is:

1. Find a collision-free path for the object (call it τ_{obj}).
2. Compute a sequence of transfer paths where the object tracks τ_{obj}.
3. Connect the transfer paths with transit paths to complete the manipulation path.

Recall that for the two-dimensional planner, the method imposes constraints on step (1) and then uses BFP to solve the constrained instance of the piano movers' problem. For this three-dimensional case, we give up the resolution-completeness for efficiency by relaxing the constraints on step (1) and by replacing BFP with RPP. Moreover, we introduce some reasonable simplifications to facilitate the incorporation of the constraints into a form easily handled by RPP. The modified constraints on step (1) for the practical planner and the associated simplifications are discussed below.

Grasp Set As in the two-dimensional case, we input a finite grasp set $\mathcal{G}$ to discretize the possible constraint submanifolds on which the transfer paths can lie. This requires $R(g, \boldsymbol{q}_{obj})$ to completely specify the system configuration for a given grasp assignment g and object configuration $\boldsymbol{q}_{obj}$. However, in this new problem we possibly have more arms in the system than are needed to manipulate the object, that is we have *working arms* and *free arms*.

Definition 4.1 Working arms *are the robot arms utilized to grasp the object* $\mathcal{M}$ *for manipulation purposes.*

Definition 4.2 Free arms *are the robot arms that are not used to manipulate* $\mathcal{M}$.

We deal with this complication by having a grasp assignment $g \in \mathcal{G}$ explicitly specify the working and free arms. For the working arms, g specifies the grasp location for the end effector of each arm and the particular inverse kinematics solution to use. For the free arms, g sets its configuration to a predetermined value where it is then treated as a fixed obstacle. This configuration is relatively non-obstructive and is the home configuration $\boldsymbol{q}_i^{home}$ associated to each robot $\mathcal{A}_i$. As in the two-dimensional case, the grasp set $\mathcal{G}$ is used in step (1) to ensure that the object path is tracing a sequence of extractable transfer paths in $\mathcal{C}_{grasp}$.

Connectable Transfer Paths In the two-dimensional case, an explicit representation of the free regions in $\mathcal{C}_{arms}$ is used during the construction of τ_{obj}. This is to verify that the generated transfer paths are connectable. Unfortunately, in the three-dimensional case, no such representation can be made since the space to store the information is beyond current capabilities in terms of memory size. Instead, we assume that there exists a transit path connecting any two collision-free arm configurations. We thus rely *heavily* on the fact that for many situations in a three-dimensional workspace this is true. Under this assumption, we can relax the constraint on step (1) to searching for a τ_{obj} that generates a sequence of *weakly-connectable* (as opposed to *connectable*) transfer paths.

Definition 4.3 *Two transfer paths τ_a and τ_b are said to be* **weakly-connectable** *by a transit path if there exists $\boldsymbol{q}^a \in \tau_a$ and $\boldsymbol{q}^b \in \tau_b$ with the same object configuration.*

Stable Configurations of $\mathcal{M}$ In the two-dimensional case, there is no restriction on where to place the object for regrasping since every object configuration is statically stable. In the three-dimensional workspace not every object configuration is a stable configuration. For example, if the arms grasping the object suddenly ungrasp, then due to gravity the object will fall. Unfortunately, incorporating the stable placements of the object into the search is not obvious. To simplify matters, the only stable placements considered for the object, where the arms are not supporting it, are the initial and goal configurations of the object (which are specified by the user). The result then, is that when the arms need to change their grasp of the object (i.e. move from one transfer path to another), they must do so while having a suitable number of the arms working to hold the object in a statically stable manner, that is the object is never placed against the obstacles. Note this fails in the case of only one robot arm in the system.[1]

Details on Grasp Assignments For a given grasp assignment g and an object configuration $\boldsymbol{q}_{obj}$, to determine whether $R(g, \boldsymbol{q}_{obj})$ is capable of holding $\mathcal{M}$ in a stable manner or is capable of moving it, requires some consideration of stability and dynamics. For simplification we further restrict the grasp assignments in $\mathcal{G}$ into two groups.

Definition 4.4 *The* **static grasp assignments** *are the grasp assignments g such that $\forall \boldsymbol{q}_{obj}$ with $R(g, \boldsymbol{q}_{obj}) \in cl(\mathcal{C}_{free})$, $R(g, \boldsymbol{q}_{obj}) \in \mathcal{C}_{stable}$.*

[1] In this case, HANDEY could be used [Lozano-Pérez et al., 1987]

Definition 4.5 *The* **motion grasp assignments** *are the grasp assignments g such that $\forall \boldsymbol{q}_{obj}$ with $R(g, \boldsymbol{q}_{obj}) \in cl(\mathcal{C}_{free})$, $R(g, \boldsymbol{q}_{obj}) \in \mathcal{C}_{grasp}$.*

Currently, we avoid the difficulty of automatically identifying these classes of grasp assignments by having it as an user input. Based on experience, the user can specify which grasp assignments are *static grasp assignments* and *motion grasp assignments*. With this simplification, the τ_{obj} we seek is simply one that generates a series of weakly-connectable transfer paths found from motion grasp assignments. For the transit paths, intermediate static grasp assignments may be needed to complete the change of grasp of $\mathcal{M}$.

4.2 Finding The Object Path

The desired object path τ_{obj} is found using RPP. Although RPP is a general planner for the piano movers' problem, we present the algorithm in the context of finding the desired object path. For a general description of RPP, we refer the reader to Chapter 7 of [Latombe, 1991]. We denote the specific algorithm for finding τ_{obj} as RPP*.

RPP* constructs τ_{obj} by first placing a fine-resolution grid over $\mathcal{C}_{obj}$, and then searching the grid using a potential field as a guide to the goal configuration. This potential field has a global minimum at the goal configuration $\boldsymbol{q}^g_{obj}$ (see Appendix C for the definition of this potential). The path is generated as a list of adjacent configurations by inserting one object configuration after the other, starting with the initial configuration of $\boldsymbol{q}^i_{obj}$.

The constraints on the object path are handled in much the same way as described in Chapter 3. Starting with $\boldsymbol{q}^i_{obj}$, RPP* finds all the motion grasp assignments $g \in \mathcal{G}$ such that $R(g, \boldsymbol{q}^i_{obj})$ is collision-free. These grasp assignments are stored as the $\mathcal{FS}$ of $\boldsymbol{q}^i_{obj}$ and also recorded in what is called the *grasp assignment list* $\mathcal{GL}$. A gradient motion from $\boldsymbol{q}^i_{obj}$ is then executed, where adjacent object configurations along the negated gradient of the potential field are traced. An adjacent configuration of $\boldsymbol{q}_{obj}$ is defined as a neighboring configuration (call it $\boldsymbol{q}'_{obj}$) in the grid with at least one grasp assignment from $\mathcal{GL}$ passing the feasibility criteria. A motion grasp assignment g is considered feasible if $R(g, \boldsymbol{q}'_{obj})$ is collision-free and the change in the configuration of the working arms (for this grasp assignment) as the object configuration goes from the $\boldsymbol{q}_{obj}$ to $\boldsymbol{q}'_{obj}$ is within a preset bound. We bound the change in the configuration of the working arms to ensure that between adjacent configurations the system remains collision-free (this is the same situation as shown in Fig. 3.3). With each new adjacent configuration $\boldsymbol{q}'_{obj}$ found along the negated gradient, the infeasible grasp assignments are removed from the grasp assignment list $\mathcal{GL}$. The updated $\mathcal{GL}$ is also stored as the $\mathcal{FS}$ of $\boldsymbol{q}'_{obj}$.

Unfortunately, prior to reaching the global minimum, an adjacent configuration along the negated gradient may fail to exist causing the gradient motion to get stuck. Let $\boldsymbol{q}_{best}$ be the last adjacent configuration found before the failure. If this failure is due to a lack of motion grasp assignments in $\mathcal{GL}$ - that is there exist grasp assignments in $\mathcal{G}$ that will clear the hold-up - then $\mathcal{GL}$ is reset to contain all the collision-free motion grasp assignments for $\boldsymbol{q}_{best}$. The gradient motion is then resumed from $\boldsymbol{q}_{best}$. Otherwise, the failure is due to the gradient motion falling into a local minima. To escape the local minima, a random walk from $\boldsymbol{q}_{best}$ is executed which traces adjacent configurations in a random manner, followed immediately by a gradient motion leading to a new local minima (or the global minimum). If the potential at the new local minima is lower than the previous one, then the escape strategy is executed from this new local minima, otherwise it is executed again from the previous one. Through this process of gradient motion, random walks, and the occasional resetting of the grasp assignment list $\mathcal{GL}$, the search moves closer to the goal. If the global minimum is attained, a path τ_{obj} of $\mathcal{M}$ described as a sequence of grid configurations and their feasible sets is returned. Because the path is composed of many random motions, a post processing step is used to smooth the motion. The smoothing algorithm is given in Appendix B.

RPP* is probabilistically resolution-complete. This means that when a τ_{obj} exists, it will find it, but the computation time may tend to infinity. The proof is given in [Barraquand and Latombe, 1991a][2]. Unfortunately, if no path exists, RPP* may run forever. Nevertheless, RPP* is usually very quick to return τ_{obj} when it exists, hence a time limit can be set beyond which it is safe to assume that no path exists.

[2]The proof is for the general RPP but it also applies in this specific case.

4.3 Extracting the Manipulation Path

The sequence of connectable transfer paths are extracted from τ_{obj} using the same greedy algorithm described in Chapter 3. The number of extracted transfer paths are minimal for the given τ_{obj}, but RPP* does not guarantee that this is the best path in that respect. In addition, it may occur that each subpath τ_{obj}^i has more that one motion grasp assignment g_i associated with it. Unlike the two-dimensional case, it may very well matter which one is used, since there is no guarantee that there exists a transit path to link the transfer paths together. In this case, choosing the correct grasp assignments to construct the transfer paths may be critical for the successful construction of the transit paths.

Definition 4.6 *A* **transit task** *specifies the initial and goal locations for which a transit path must be found to connect them.*

The subpaths τ_{obj}^i and their grasp assignments g_i can be organized into successive layers, as illustrated in Fig. 4.1. Each layer contains all the transfer paths generated for the same subpath of τ_{obj}; the transfer paths differ by the grasp assignment. Selecting one such path in every layer yields a series of transit tasks: the first consists of achieving the initial grasp of the object from the $\boldsymbol{q}_{sys}^i$; it is followed by a possibly empty series of transit tasks to change grasps between two connectable transfer paths; the last transit task is to achieve the goal system configuration. Hence, it remains to identify a grasp assignment in each layer of the graph shown in Fig. 4.1, such that there exist transit paths accomplishing the corresponding transit tasks.

Assume without loss of generality that all arms are initially at their non-obstructive configurations. Our planner first chooses a transfer path (anyone) in the first layer. Consider the transit task of going from the initial system configuration to the configuration where the arms achieve the grasp assignment specified in the chosen transfer path, with $\mathcal{M}$ being at its initial configuration. The coordinated path of the arms is generated using RPP, that is the general randomized path planner. If this fails, a new attempt is made with another transfer path in the first layer; otherwise, a transfer path is selected in the second layer. The connection of the system configuration at the

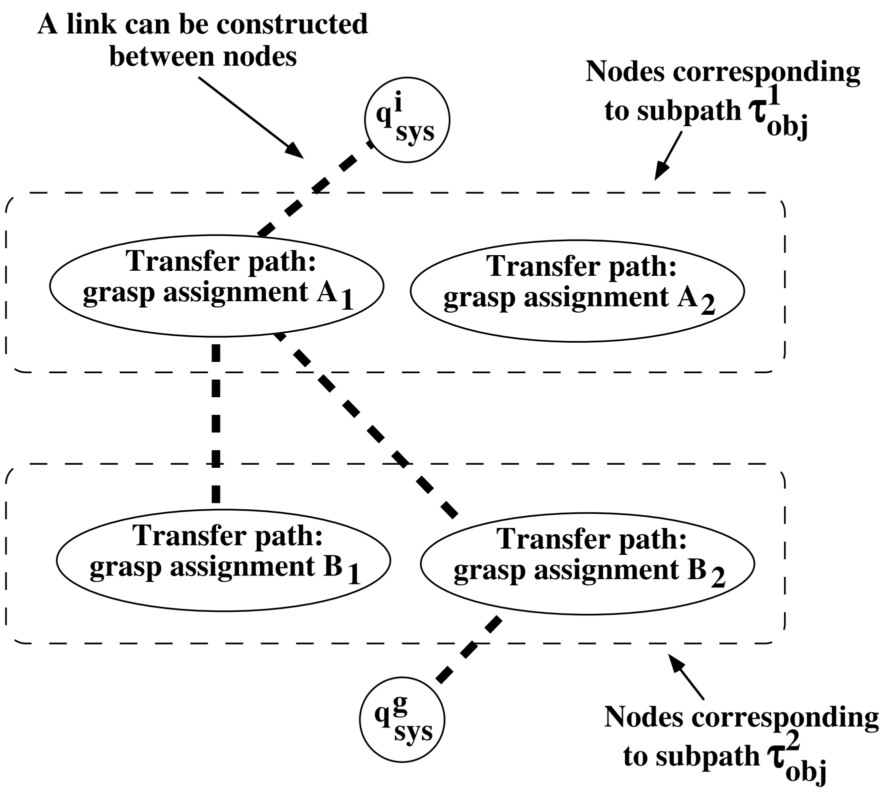


Figure 4.1: The layered graph.

end of the first transfer path to the system configuration at the start of this second transfer path forms a new transit task.

The transit task between two transfer paths is more difficult to solve. To understand the difficulty, imagine the case where $\mathcal{M}$ is a long bar requiring two arms to move. Consider the situation where the bar is grasped at its two ends and the regrasp requires swapping the grasp location of the two arms. This regrasp is not possible without introducing an intermediate grasp. We illustrate this example in Fig. 4.2. After grasping and rotating the bar a change of grasp is required (sequence (a) to (d) in Fig. 4.2); arm 1 ungrasps one end of the bar and regrasps it at its center while arm 2 continues to hold the bar without moving (sequence (d) to (f)); then arm 2 ungrasps the bar and regrasps it at the other end (sequence (f) to (h)); finally, arm 1 ungrasps the center of the bar and regrasps it at its free end, thus completing the change of grasp (sequence (h) to (i)).

We address this difficulty by breaking the transit task between connectable transfer paths into smaller transit subtasks. Each transit subtask consists of going from one grasp assignment (at least a static grasp assignment) to another in such a way that no two arms use the same grasp location at the same time. In this process, we allow all the arms to be used. We start with the first grasp assignment and generate all the potential grasp assignments that may be achievable from it (assuming the corresponding transit paths exist). We generate the successors of these new assignments, and so on until we reach the desired assignment (the one used in the next transfer task). For each sequence that achieves this desired assignment, we test that it is actually feasible by using RPP to generate a transit path between every two successive grasp assignments. We stop as soon as we obtain a feasible sequence. The concatenation of the corresponding sequence of transit paths forms the transit path linking the two connectable transfer paths. We then proceed to link to the next layer of transfer paths.

When we reach a transfer path in the last layer, its connection to the goal system configuration is carried out in the same way as the connection of the initial system configuration to the first layer.

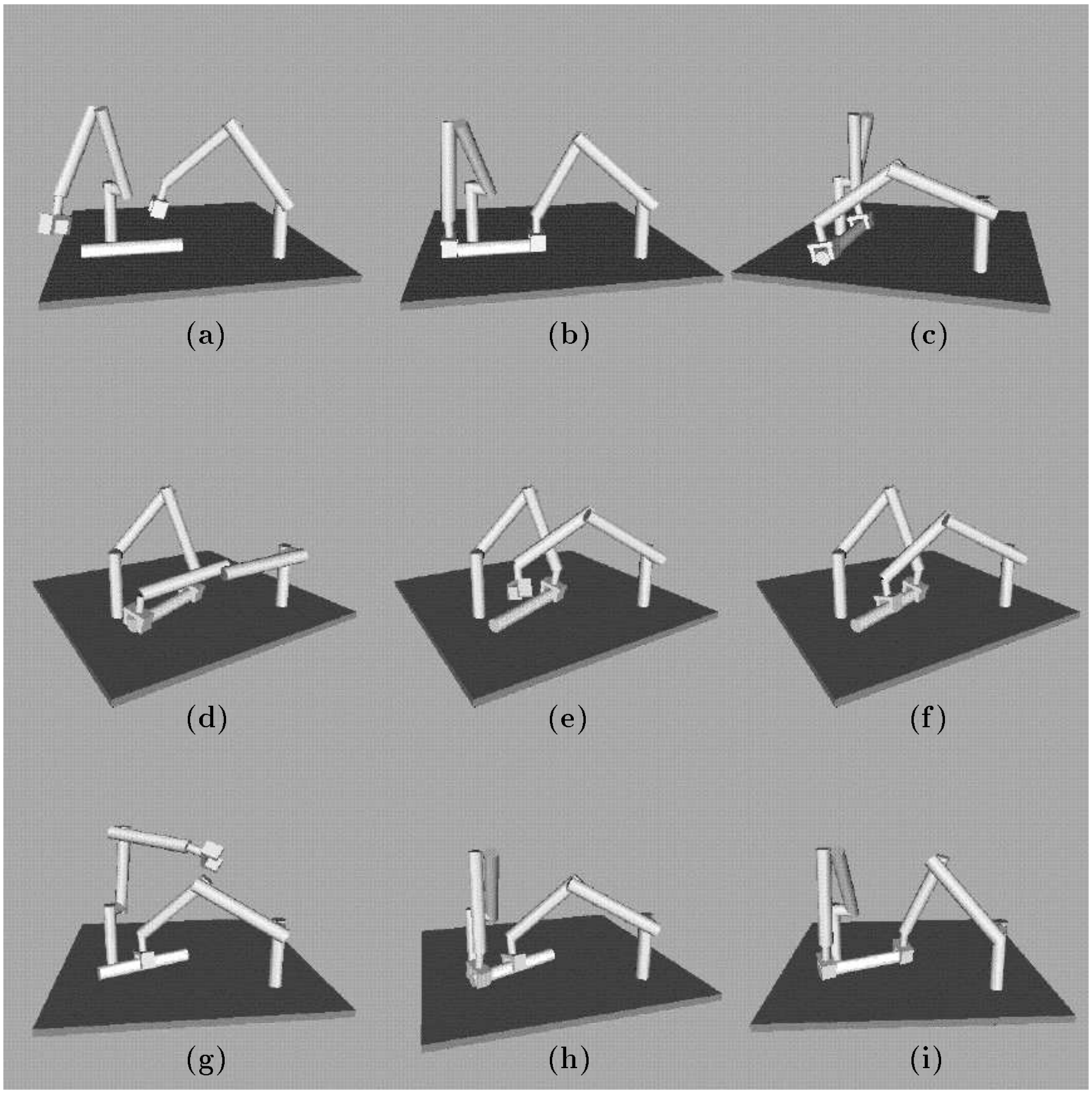


Figure 4.2: An example illustrating the complexity of changing grasps.

The resulting sequence of transit and transfer paths is the manipulation path connecting $\boldsymbol{q}^i_{sys}$ to $\boldsymbol{q}^g_{sys}$.

4.4 Sliding Grasps

In the algorithms presented thus far, the robots are restricted to grasp the object in a rigid fashion. In some manipulation situations, it may be useful to relax this constraint and allow the object to slip within the grasp in a controllable manner. We now consider this case in an ad hoc approach. We restrict sliding to the multi-grasp case (i.e. two or more sets of grippers act on the object for dextrous manipulation), and assume that any grasp can be made to slide on the object, when it is kinematically feasible. We completely ignore the many important issues related to the actual control of the slip as studied in [Kao and Cutkosky, 1992][Tremblay and Cutkosky, 1993].

Let $\mathcal{G}'$ be a subset of $\mathcal{G}$ which contains motion grasp assignments and such that all $g \in \mathcal{G}'$ have at least two working arms. To facilitate the sliding of the object within a grasp, we define the notion of neighbouring grasps.

Definition 4.7 *Two elements in $\mathcal{G}'$ are said to be* **neighbouring grasp assignments** *if they specify the same working arms, the change in position and orientation between grasp locations for each arm is within some predefined threshold, and the net torque on the object to realize the sliding grasp is zero.*

Furthermore, for a given $\boldsymbol{q}_{obj}$ and neighbouring grasp assignments g_1 and g_2, the working arms move from $R(g_1, \boldsymbol{q}_{obj})$ to $R(g_2, \boldsymbol{q}_{obj})$ by having the end effectors slide accordingly, that is the UNGRASP/GRASP operations are not invoked. We make the assumption that, if $R(g_1, \boldsymbol{q}_{obj})$ and $R(g_2, \boldsymbol{q}_{obj})$ are collision free and the change in configuration of the working arms is within some preset bound, then they lie in the same connected component of $\mathcal{C}_{grasp}$. Note that this is the same sort of assumption we use in constructing the connected components of $\mathcal{CF}_{arms}$ in Chapter 3. Thus, transfer paths may now consist of motions where the object slides within the grasp of the grippers.

We incorporate the sliding grasps by modifying the search for τ_{obj}. The basic idea is to augment $\mathcal{GL}$ with the neighbouring grasp assignments of those already in $\mathcal{GL}$. Consequently, object configurations that previously were not reachable because it required a sliding grasp can now be achieved.

More specifically, let $\mathcal{GL}'$ be the set of neighbouring grasp assignments of those in $\mathcal{GL}$. RPP* is modified to also consider $\mathcal{GL}'$ for each adjacent configuration encountered in the search. For a given adjacent configuration $\boldsymbol{q}'_{obj}$ RPP* filters out of $\mathcal{GL}'$ those grasp assignments g that do not yield a collision-free $R(g, \boldsymbol{q}'_{obj})$. In addition, for g_1 in the updated $\mathcal{GL}'$ and its neighbouring grasp assignment g_2 in $\mathcal{GL}$, if the change in configuration of the working between $R(g_1, \boldsymbol{q}'_{obj})$ and $R(g_2, \boldsymbol{q}'_{obj})$ is greater than the preset bound, then g_1 is removed from $\mathcal{GL}'$. $\mathcal{GL}$ is augmented with $\mathcal{GL}'$. The updated $\mathcal{GL}$ is also stored as the $\mathcal{FS}$ of $\boldsymbol{q}'_{obj}$. The search proceeds accordingly.

In extracting the transfer paths, if two weakly-connectable transfer paths have neighbouring grasp assignments, then they are glued into one transfer path by inserting a sliding motion between the two.

4.5 Incompleteness

The step to find τ_{obj} is probabilistically resolution-complete. Given the sequence of weakly-connectable transfer paths, the step to find the transit paths is also probabilistically resolution-complete. This is because we use RPP. However, the combination of the two steps yields an incomplete algorithm. This means that even if a manipulation path exists, it may not find it, and if a solution does not exist, it cannot report failure.

The reason is because the transfer paths extracted from τ_{obj} are only weakly-connectable. Indeed, if there does not exist a transit path to link two weakly-connected transfer paths, RPP will not report failure, and consequently, other sequences of transfer path will not be considered.

Despite this limitation various experiments with the planner demonstrate that it is quite efficient and reliable in finding manipulation paths when they exist. Therefore, if a path is not found by time T, where T is determined empirically, then a path is unlikely to be found. This is the failure condition for the algorithm.

4.6 Results and Discussion

We have implemented a multi-arm manipulation planner based on the presented algorithm. It is written in C and runs on a DEC Alpha workstation under UNIX. We have applied the planner to three different situations to show its reliability. The first example is for a system of three general 6R robot arms whose kinematics are such that some interesting motions can be found. The second example is for three Puma 560 arms, modelled after the setup found in the Computer Science Robotics Laboratory, Stanford University. And finally, we show the planner applied to generating realistic human arm motions.

For these examples, in computing the transit paths, RPP uses the sum of the angular joint distances to the goal configuration as the guiding potential. In finding both the transit paths and τ_{obj}, we limit the amount of computation spent in RPP and RPP* to three backtrack operations (see Appendix C), after which the planner returns failure. Failure to find τ_{obj} results in the immediate failure to find a manipulation path. Similarly, a failure to find transit paths to link together the layers of transfer paths results in a failure to find a manipulation path. The time for the planner to report failure depends on the problem, with some examples ranging from 30 seconds to a few minutes. To check collisions we use Quinlan's algorithm - a fast hierachical collision checking algorithm [Quinlan, 1994].

4.6.1 Three General 6R Robot Arms

The system consists of three robot arms, each with six revolute joints, working in a three dimensional workspace. The non-obstructive configuration ($\boldsymbol{q}_i^{home}$) of each arm is one where the arm stands vertical.

Fig. 4.3 shows a manipulation path generated by the planner for an L-shaped object. This object requires two arms to move it. The object is taken through the window in the large obstacle located in the middle of the workspace. Notice that in the

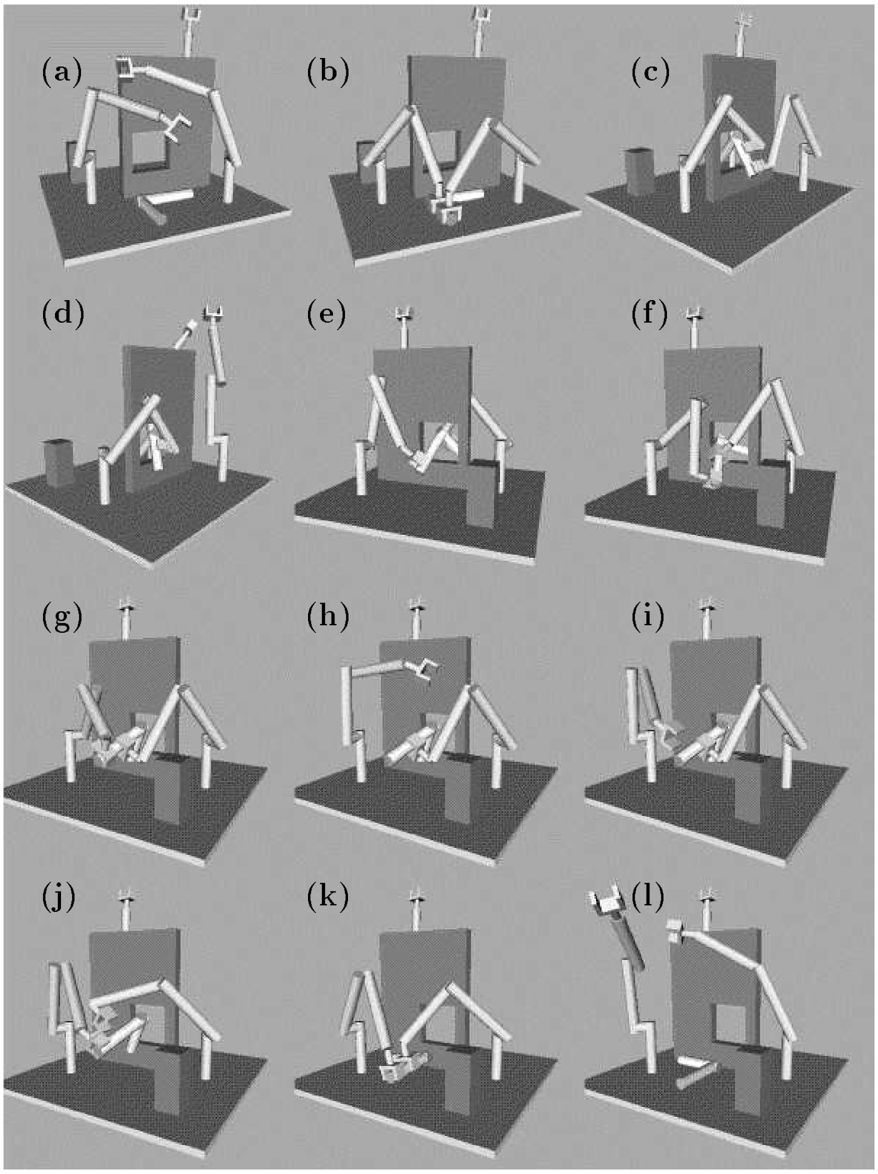


Figure 4.3: Manipulating a L-shaped object.

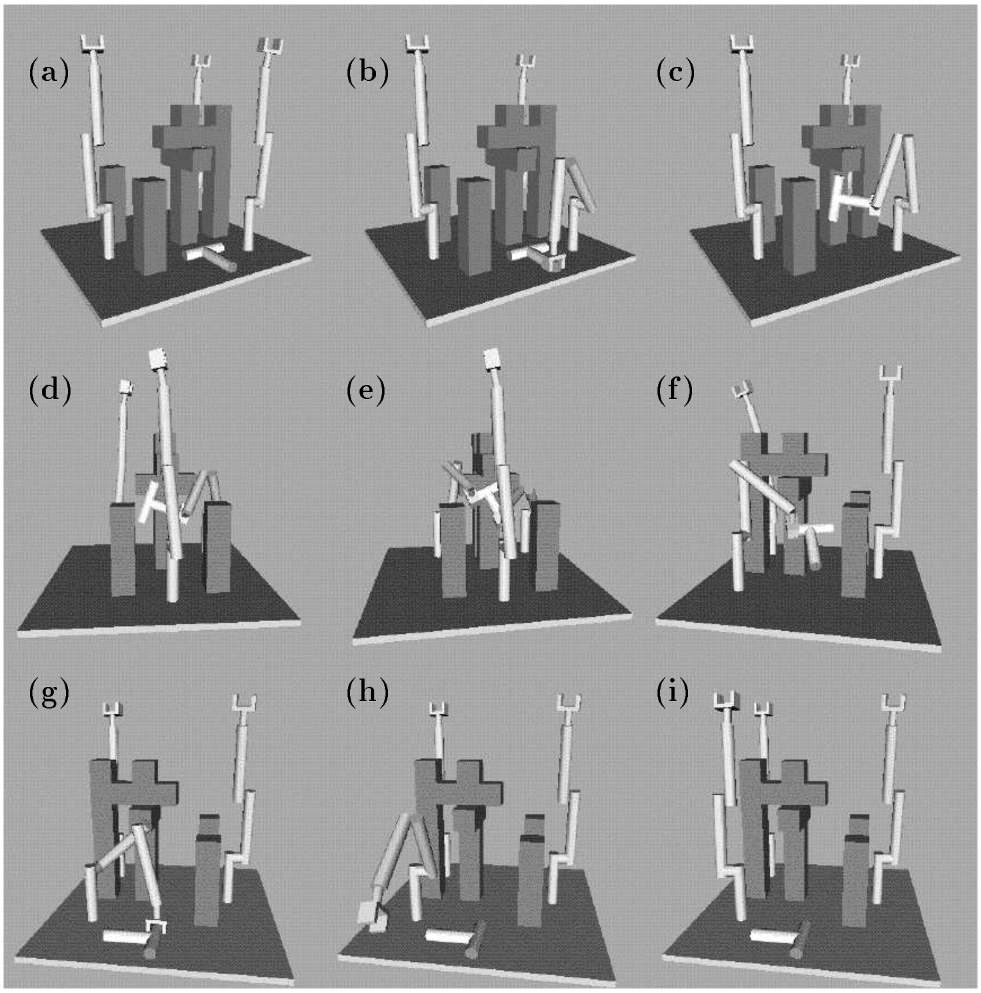


Figure 4.4: Manipulating a T-shaped object.

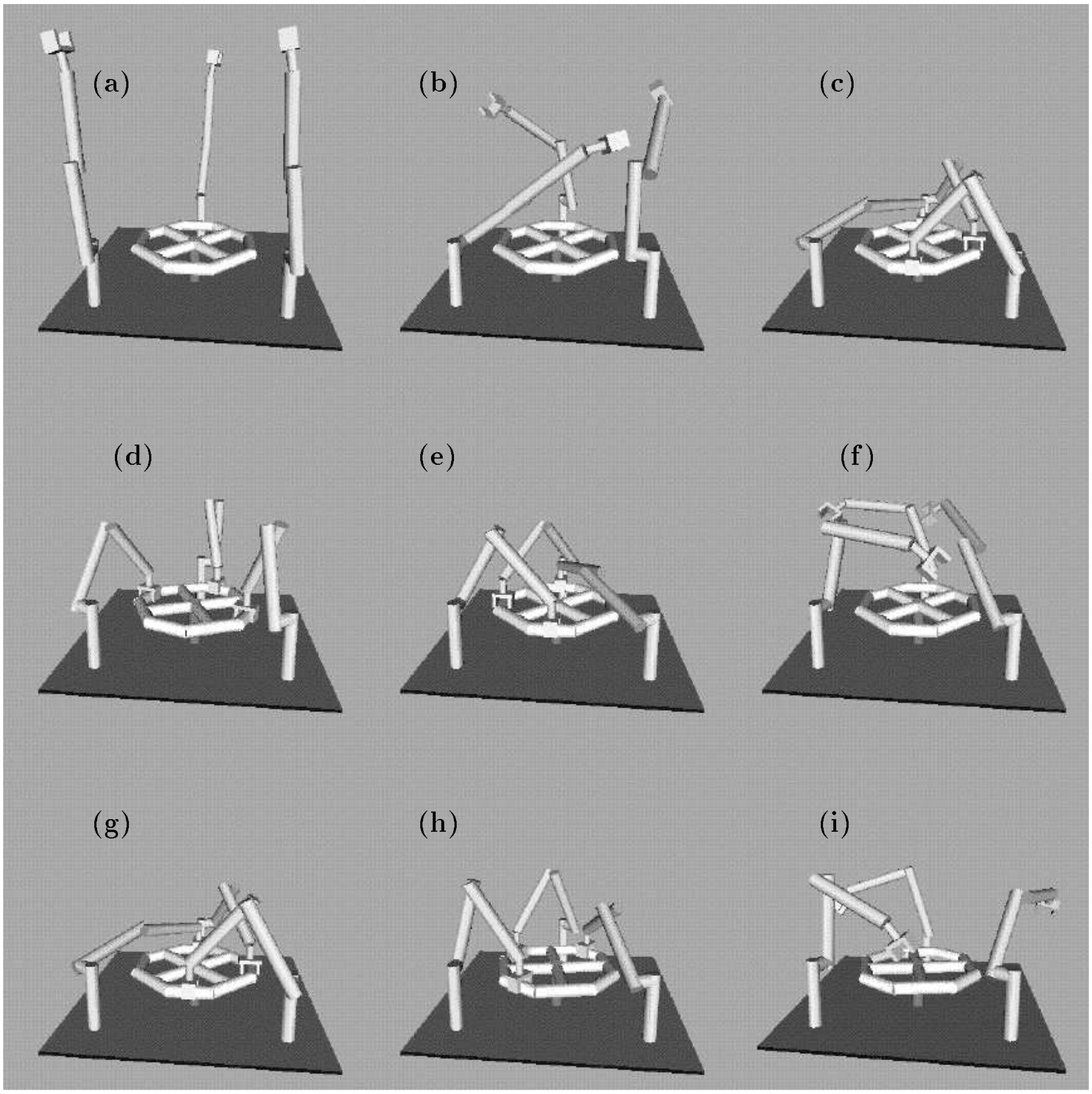


Figure 4.5: Manipulating a wheel.

change of grasp, at least one arm is holding the object at all times. For this motion, it took 45 seconds to find the object path and 30 additional seconds to complete the manipulation path. For the generation of τ_{obj}, each axis of the object's C-space was discretized into 100 units. For the generation of the transit paths, the joint angles of the arms were discretized into intervals of 0.05 radians. There are 64 grasp locations on the object, yielding a grasp set of around 25,000 elements.

Fig. 4.2 shows a manipulation path generated for a long bar requiring two arms to move it. This example illustrates the complexity of changing grasps. For this motion, it took 25 seconds to find the object path and an additional 20 seconds to complete the manipulation path. The same discretizations as above were used. There are 24 grasp locations of the long bar, yielding a grasp set of around 10,000 elements.

Fig. 4.4 shows a manipulation path found for a T-shaped object. This object requires a single arm to move it, and only two arms are used along the computed path. One arm first grasps the object at one end of the T. It passes the object to another arm that grasps it at its other end and brings it to its goal configuration. For this motion, the planner took 40 seconds to find the object path and then another 25 seconds to complete the manipulation path. The same discretizations as above were used. There are 49 grasp locations on the T-shaped object, yielding a grasp set of around 1,200 elements.

Fig. 4.5 shows a manipulation path found for rotating a giant wheel one and a quarter turns. The wheel requires all three arms to rotate it. Notice that to complete the desired rotation an intermediate ungrasp/regrasp operation is needed. For this motion, the planner took 30 seconds to find the object path and then another 25 seconds to complete the manipulation path. The same discretizations as above were used. There are 4 grasp locations on the T-shaped object, yielding a grasp set of around 3,100 elements. Here we specify a static grasp assignment that requires no arms, that is the wheel is always in a stable configuration without the arms having to hold it.

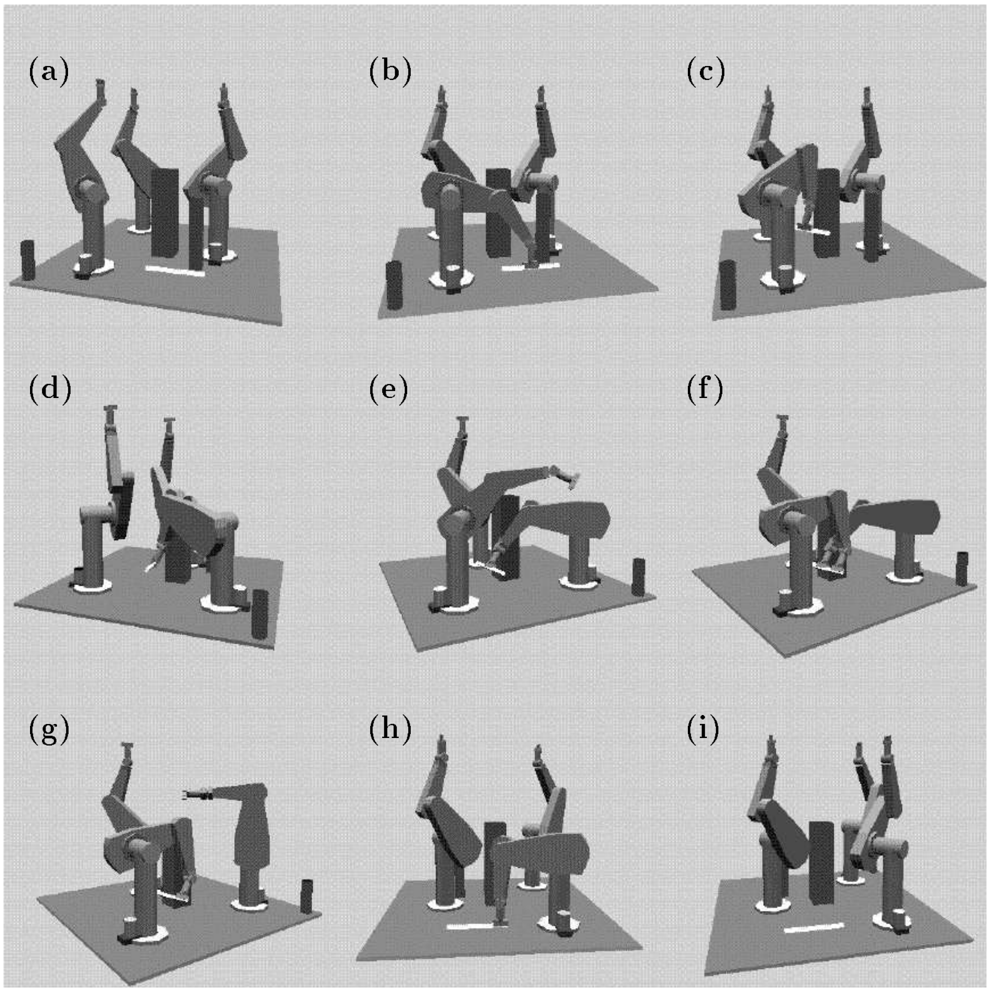


Figure 4.6: Puma example: light object.

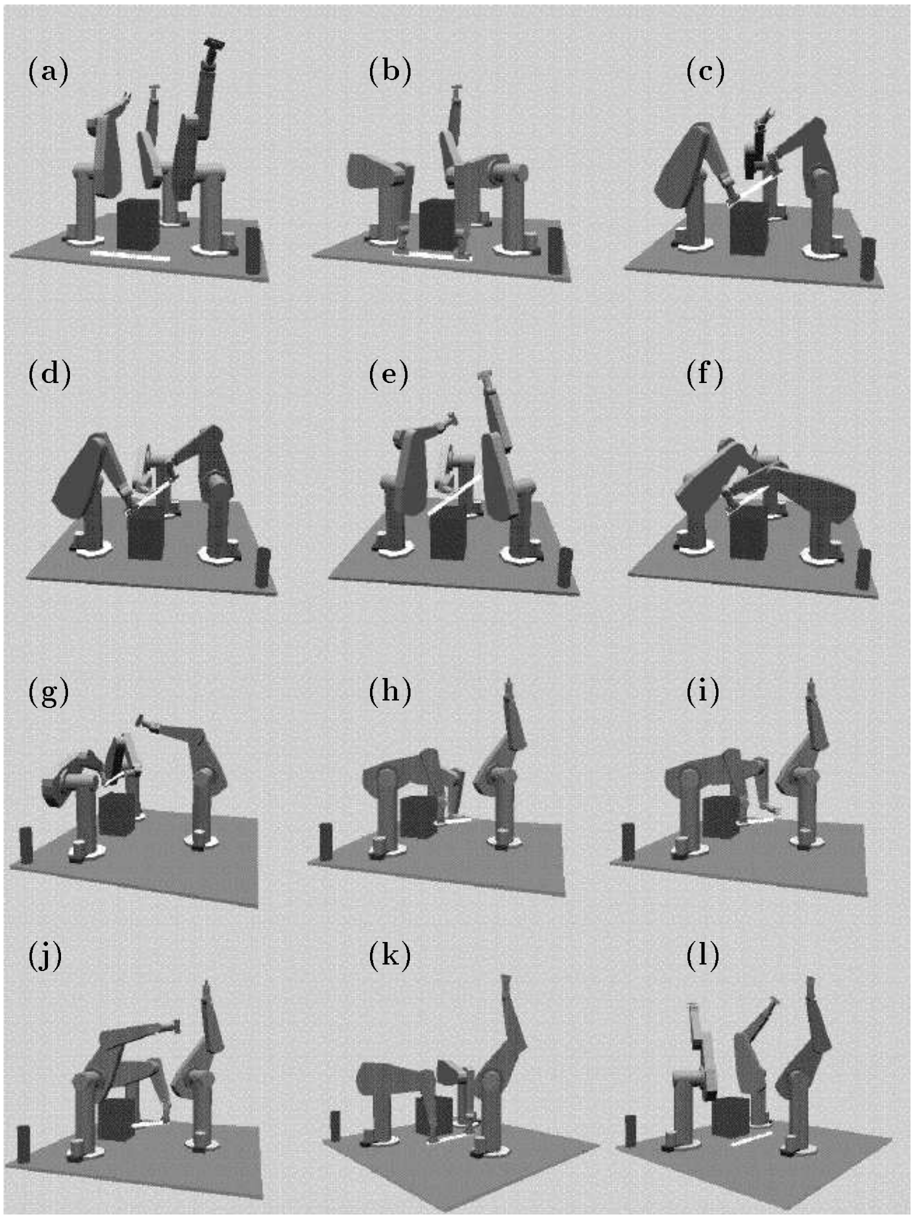


Figure 4.7: Puma example: heavy object.

4.6.2 Three PUMA 560 Arms

We consider an actual robotics workcell by modelling the three PUMA arms found in the Computer Science Robotics Laboratory, Stanford University. The non-obstructive configuration ($\boldsymbol{q}_i^{home}$) of each arm is one where the arm stands vertical.

Fig. 4.6 shows a manipulation path generated by the planner for a light, rectangular shaped object requiring only one arm for manipulation. The task is to take the object to the other side of the workspace. Notice that to complete the task, the first arm hands off the object to the second arm (at snapshot (f) of Fig. 4.6). For this motion, it took three minutes to find the object path and an additional two minutes to complete the manipulation path. For the generation of τ_{obj}, each axis of the object's C-space was discretized into 100 units. For the generation of the transit paths, the joint angles of the arms were discretized into intervals of 0.05 radians. There are 12 grasp locations on the object, yielding a grasp set of 288 elements. The cylinder in the corner of the workspace should aid in viewing the series of snap shots.

Fig. 4.7 shows a manipulation path generated by the planner for a heavy, rectangular shaped object. The object is light enough for one arm to hold it statically, but requires two arms to manipulate it. The task is to take the object to the other side of the box and to rotate it 180 degrees. Half way through the manipulation, the grasping arms regrasp the object to complete the desired rotation. The third arm moves to hold the object statically as the other arms regrasps (sequence (c) to (f)). For this motion, it took around three minutes to find the object path and an additional three minutes to complete the manipulation path. The same discretization as the one arm case was used. There are 12 grasp locations on the object, yielding a grasp set of around 5,000 elements.

4.6.3 Human Arm Manipulation

To demonstrate the variety of different manipulators that the planner can handle, we show some examples where a seated human figure and a robot arm execute manipulation tasks. These examples are taken from [Koga et al., 1994]. Each human arm has seven degrees of freedom, plus an additional nineteen degrees of freedom for the hand.

The non-obstructive configurations for the human arms are ones in which they are held out to the side. The robot has six revolute joints and three degrees of freedom for the fingers of the end-effector. The non-obstructive configuration for the robot is one in which it stands vertically. The inverse kinematics of the human arms come from Kondo's algorithm [Kondo, 1994] where the redundant degree of freedom is resolved using the *sensori-motor transformation model* derived from neurophysiological experiments [Soechting and Flanders, 1989]. A detailed description of the algorithm is given in [Kondo, 1994]. In addition, we seat the human on a swivel chair. By adding this extra degree of freedom, we allow the arms to access a greater region, and hence tackle more interesting manipulation tasks. The rotation of the chair tracks the object, to keep it essentially in an optimal position with respect to the workspace of the arms [McCormick, 1982]. The dimension of the composite configuration space in these examples is seventy eight.

Fig. 4.8 shows a path generated by the planner for the human arms to bring the glasses on the table to the head of the human figure. We specify that both arms should be used to manipulate the glasses (this is defined in the grasp set). Notice that during the regrasping phase, at least one arm is holding the glasses at all times. We consider sliding grasps in this example. For this motion it took one minute to find the object path and an additional two minutes to complete the manipulation path. For the generation of τ_{obj}, each axis of the object's C-space was discretized into 100 units. For the generation of the transit paths, the joint angles of the arms were discretized into intervals of 0.05 radians. There are 70 grasp locations on the glasses, yielding a grasp set of 424 elements.

Fig. 4.9 shows a path generated by the planner for the human arms and the robot arm cooperating to manipulate a chess box. Having the different arms working together presents no difficulty to our planning approach. The planner simply needs to know the correct inverse kinematics algorithm to apply to each arm. For this example, in defining the grasp set, we specify two classes of grasps, one in which all three arms are used, and another in which only the two human arms are utilized. Also, the human grasps are allowed to slide. For this motion it took about one and a half minutes to find the object path and an additional two minutes to complete the

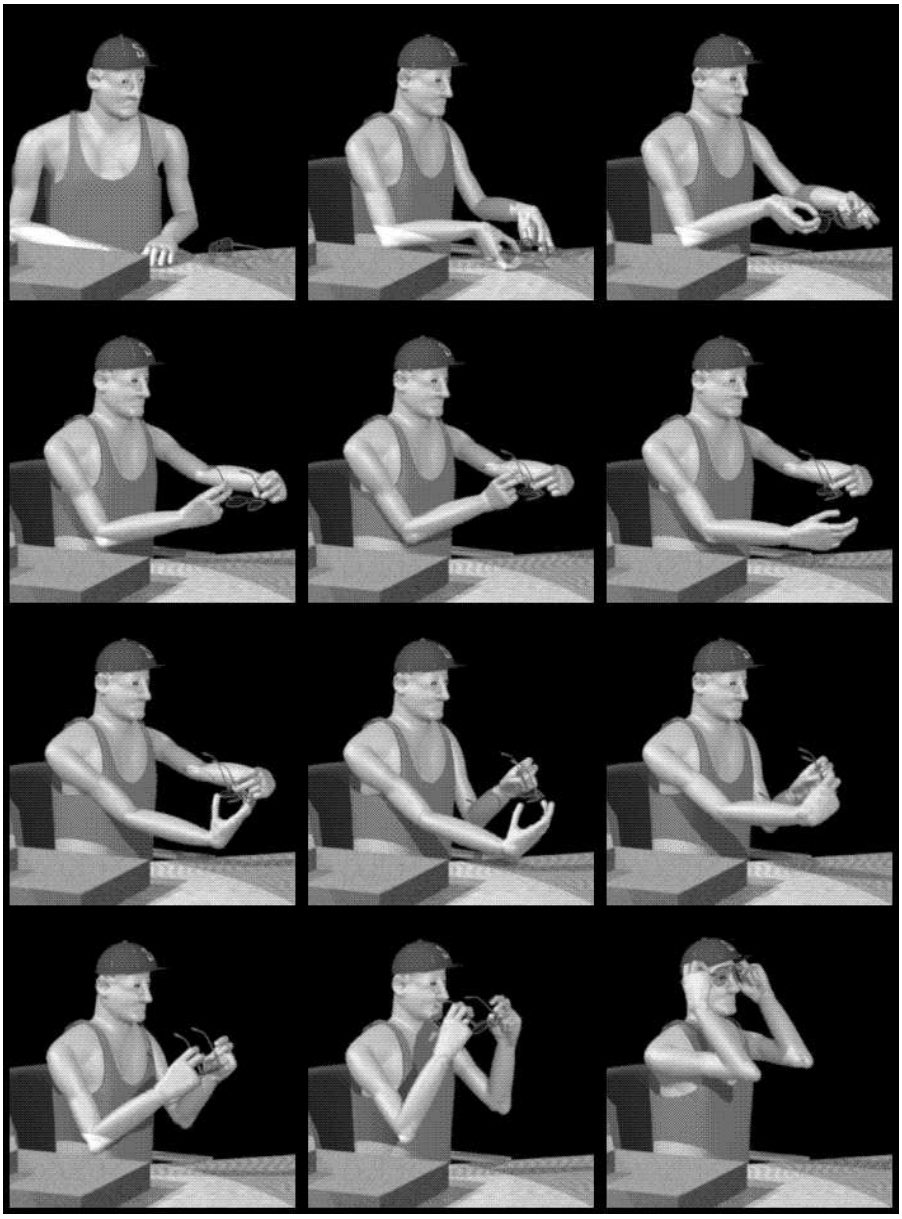


Figure 4.8: Manipulating a pair of glasses.

manipulation path. The same discretizations as above were used. There are 40 grasp locations on the box yielding a grasp set of around 2,600 elements.

In addition, the planner was used to automatically generate a rather complex computer animation clip [Kuffner et al., 1994]. The motions consist of a human figure putting on his glasses, moving a box, lifting a robot out of the box, and then playing chess with the robot. The major motions in the clip were computed by specifying only ten intermediate systems configurations.

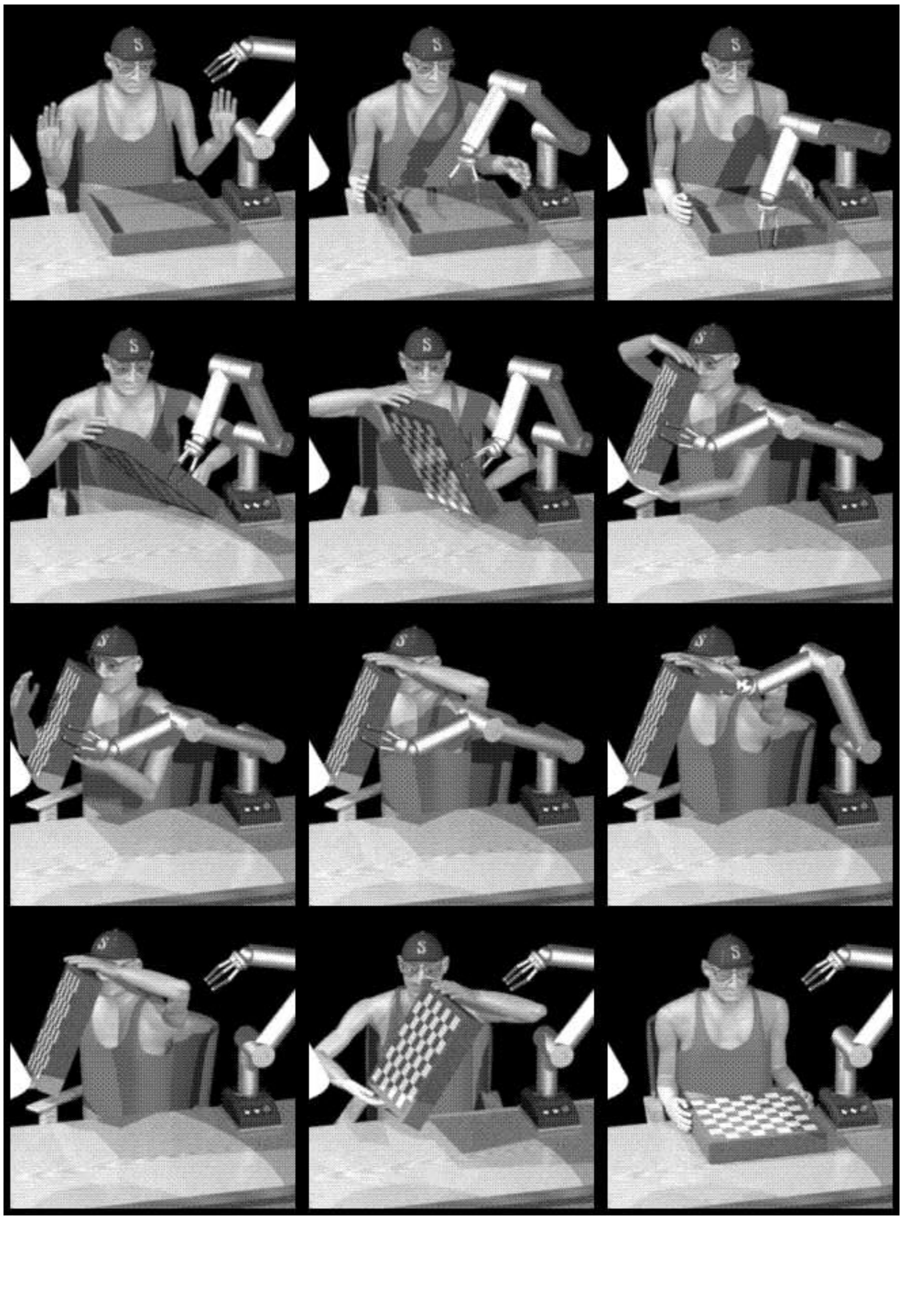


Figure 4.9: Cooperative motion between a human and a robot.

Chapter 5

Dynamic Constraints and Trajectory Planning

The manipulation planners presented in Chapters 3 and 4 deal with the first step of our two phase approach to trajectory planning. This chapter deals with the second step; we compute a time history of the position, velocity and acceleration for each robot along the given path, while satisfying the dynamic constraints imposed by the system (e.g. actuator torque limits). This is done by utilizing a time parameterizing scheme that considers the dynamic models of the robots, their actuator limits, the control strategy, and the friction interaction between the gripper and the movable object (hereafter we call the movable object the payload). In fact, with these models we determine the minimum-time parameterization for these manipulation paths. This is the path-constrained-optimal-control problem for robots.

Specifically, we consider the minimum-time parameterization for three types of paths:

- transit paths (open-kinematic-chain);
- transfer paths with one robot arm (open-kinematic-chain);
- transfer paths with cooperating robot arms (closed-kinematic-chain).

These three cases cover the possible combinations of robots and payload during a manipulation path. We only consider rigid grasps.

By classifying the first case (transit path) as the basic problem, the other two are the basic problem with added constraints. The second case of the single arm transferring a payload has the additional constraint that the payload not slip out of the grasp of the gripper. The third case has the additional constraint that the optimization deal with the redundant actuation in the closed-chain motion in a fashion compatible with the multi-arm control strategy for the real robots. Incorporating these additional constraints into the basic problem is the focus of this chapter.

5.1 The Open-Chain Robot

There exists a well established path-constrained-optimal control algorithm for open-chain robots [Bobrow et al., 1985][Shin and Mackay, 1985][Pfeiffer and Johanni, 1987][Shiller and Lu, 1990]. We use this algortithm as the foundation for finding the minimum-time parameterization for the other two cases (the transfer paths). We give a brief summary of the algorithm, and particulary concentrate on the portions relevant for developing the algorithms for the other two cases.

Consider the situation where a robot is prescribed to follow a specified path. The generalized coordinates of the arm are then parameterized in terms of a single variable s, where s can be regarded as the distance travelled along the path. The optimal control problem is to find the time parameterization $s = s(t)$ that minimizes the travel time while satisfying the limits on the actuators.

These dynamic constraints are represented using the equations of motion for the robot. For an n degrees of freedom open-chain robot the equations of motion can be written as,

$$A(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{b}(\dot{\boldsymbol{q}}, \boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{5.1}$$

where A is the $n \times n$ inertia matrix, $\boldsymbol{b}$ is the $n \times 1$ vector of velocity dependent forces, $\boldsymbol{g}$ is the $n \times 1$ vector of the gravity forces, $\boldsymbol{\tau}$ is the $n \times 1$ vector of actuator torques, and $\boldsymbol{q}, \dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the generalized coordinates, velocities, and accelerations, respectively. Furthermore, because the path is constrained, the equations of motion become parameterized by s and the vector of actuator torques become,

$$\boldsymbol{\tau} = \boldsymbol{a}\ddot{s} + \boldsymbol{b}'\dot{s}^2 + \boldsymbol{g} \tag{5.2}$$

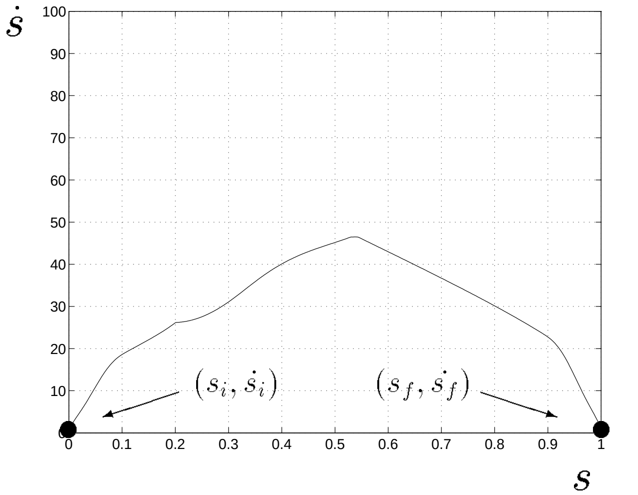


Figure 5.1: Velocity profile in the $s - \dot{s}$ plane.

$$\boldsymbol{a} = A\frac{d\boldsymbol{q}}{ds}$$

$$\boldsymbol{b}' = A\frac{d^2\boldsymbol{q}}{ds^2} + \tilde{\boldsymbol{b}}$$

where $\dot{s}$ and $\ddot{s}$ are a measure of the velocity and acceleration of the robot along the path respectively, and $\tilde{\boldsymbol{b}}$ is the vector $\boldsymbol{b}$ with the path velocity $\dot{s}^2$ factored out (it can be shown that each vector element of $\boldsymbol{b}$ has an $\dot{s}^2$ term). The bounds on the actuators are,

$$\boldsymbol{\tau}_{min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max}. \tag{5.3}$$

To find the minimum-time parameterization $s(t)$, the *two point boundary value problem* (TPBVP) with boundary constraints $(s_i, \dot{s_i})$ and $(s_f, \dot{s_f})$ is considered, where $(s_i, \dot{s_i})$ and $(s_f, \dot{s_f})$ are the initial and final states of the robot respectively. In the s - $\dot{s}$ plane (see Fig. 5.1), this translates to finding a curve that connects the two states while satisfying Eq. (5.2) and the actuator torque bounds given by (5.3). Furthermore, since the travel time is

$$t_f = \int_{s_o}^{s_f} \frac{ds}{\dot{s}} \tag{5.4}$$

by maximizing $\dot{s}$ we minimize the travel time. Hence, the curve we seek in the s - $\dot{s}$ plane is one that solves the TPBVP and maximizes the area it encloses.

Bobrow et al. show in [Bobrow et al., 1985] that the optimal curve has $\ddot{s}$ at its maximum possible acceleration or deceleration at all times (Shiller and Lu show that there exist some special paths with *critical arcs* where this is not true [Shiller and Lu, 1990] - we assume our paths are without the critical arcs). This reduces the problem to finding the switching points from maximum acceleration to maximum deceleration while satisfying the aforementioned constraints. An efficient and robust method for finding these switching points is given in [Shiller and Lu, 1990].

The algorithm for finding the switching points makes use of what is called the *velocity limit curve* and the maximum and minimum $\ddot{s}$ for a given s and $\dot{s}$. The limit curve is the locus of the maximum instantaneous $\dot{s}$ along the path and serves as an upper bound for the optimal curve (see Fig. 5.2). The important point is that regardless of the robot system (i.e. any one of the three cases we consider), the switching point algorithm will return the optimal curve if these input values are provided. Thus, from our viewpoint the key is the ability to compute the required input information.

For the open-chain robot, these input values are found by noting that for a given s we have a linear programming problem with variables $\dot{s}^2$, $\ddot{s}$, and $\boldsymbol{\tau}$. This was first noted in [McCarthy and Bobrow, 1992]. The details to transform the above problem into a linear programming problem is given in Appendix D. Using the Simplex linear programming algorithm, the desired input for the switching point algorithm can be found, resulting in the minimum-time parameterization of the transit paths.

For this particular case of the open-chain robot, an elegant graphical method can be employed instead of the Simplex algorithm. We refer the reader to [Pfeiffer and Johanni, 1987] for details.

5.2 The Open-Chain Robot Carrying a Payload

The simplest strategy for dealing with an open-chain robot carrying a payload is to model the payload as rigidly attached to the manipulator. We can then apply the same method as in Section 5.1. However, in the actual execution the inertial effects on the payload may exceed the holding capacity of the gripper, resulting in the payload

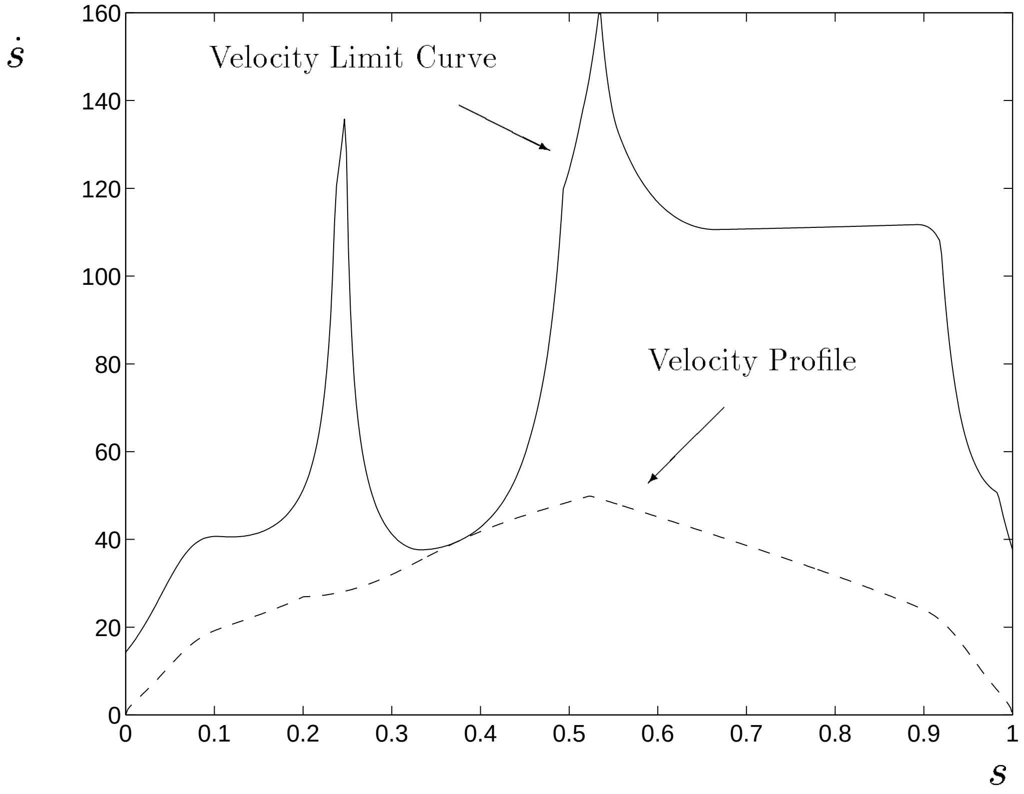


Figure 5.2: Velocity profile and the limit curve.

flying out of the grasp. We need a *no-slip* constraint. In the following analysis, we consider the generic case of parallel jaw grippers.

Friction is what maintains the rigid grasp of the payload. If the interaction forces between the gripper and the payload are within the friction limits, then the rigid grasp is maintained. This section discusses the approach taken to add this consideration of friction limits into the minimum-time parameterization algorithm.

We consider first, an appropriate friction model. We use the viscoelastic rubber friction model developed by Howe, Kao, and Cutkosky [Howe et al., 1990]. We justify the use of such a model by the fact that most robot finger tips have a rubber surface. From experiments where they measure the point of slip for a soft rubber planar surface under combined torsion and shear loading, they give the following conservative friction model,

$$f_t \leq \mu|f_n| - \alpha|m_n| \tag{5.5}$$

where f_t is the shear loading, m_n is the torsional loading, f_n is the normal force, α is a "proportionality constant", and μ is the coefficient of friction. α is equal to the ratio of the maximum measured torsion to the maximum measured shear. As long as the constraint in (5.5) is satisfied, the contact will remain fixed.

The shear and torsional loading are related to the interaction forces between the gripper and the payload by,

$$m_z = m_n \tag{5.6}$$

and

$$\sqrt{f_x^2 + f_y^2} = f_t, \tag{5.7}$$

where the interaction forces are represented by the vector,

$$\boldsymbol{\eta}^{int} = \begin{bmatrix} f_x \\ f_y \\ f_z \\ m_x \\ m_y \\ m_z \end{bmatrix} \tag{5.8}$$

acting at the grasp contact center. The x and y components lie in the contact plane, and the z component lies perpendicularly to it. Thus by Eqs. (5.5), (5.6), and (5.7) we can bound the interaction forces such that a rigid grasp is maintained.

We further approximate this bound to:

$$|f_x| \leq 0.707(\mu(2f_g + |f_z|) - \alpha|m_z|) \tag{5.9}$$

$$|f_y| \leq 0.707(\mu(2f_g + |f_z|) - \alpha|m_z|)$$

$$|m_z| \leq \frac{\mu(2f_g + |f_z|)}{\alpha},$$

where f_g is the gripping force. Note that for the component of force normal to the contact plane we use $2f_g + |f_z|$. The $2f_g$ component comes from the fact that there are two contact planes in the grasp, while the $|f_z|$ component is due to the inertial and gravity effects.

To bring into evidence $\boldsymbol{\eta}^{int}$, we break up the system into two pieces. We write the equations of motion for the payload where the interaction forces become the active forces on the payload, and the equations of motion for the arm where the interaction forces become the external forces acting on the arm.

To the payload, a coordinate frame is attached at the grasp contact center such that the z-axis of the frame lies perpendicularly to the contact plane. The position and orientation of this frame with respect to some inertial reference frame forms a set of generalized coordinates for the payload. The equations of motion for the payload is then,

$$\Lambda_{obj}(\boldsymbol{x})\ddot{\boldsymbol{x}} + \boldsymbol{\pi}_{obj}(\dot{\boldsymbol{x}}, \boldsymbol{x}) + \boldsymbol{p}_{obj}(\boldsymbol{x}) = \boldsymbol{\eta}^{int} \tag{5.10}$$

where Λ_{obj} is the 6×6 inertia matrix, $\boldsymbol{\pi}_{obj}$ is the 6×1 vector of velocity dependent terms, $\boldsymbol{p}_{obj}$ is the 6×1 vector of gravity dependent terms, and $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, and $\ddot{\boldsymbol{x}}$ are the generalized coordinates, velocities, and accelerations respectively.

When there is no-slipping the payload is rigidly attached to the arm, thus,

$$\boldsymbol{x} = T(\boldsymbol{q}) \tag{5.11}$$

where T maps the joint coordinates of the arm to the position and orientation of the coordinate frame attached at the grasp contact center (forward kinematics). Furthermore, using the definition of the Jacobian we get,

$$\dot{\boldsymbol{x}} = J\dot{\boldsymbol{q}} \tag{5.12}$$

and

$$\ddot{\boldsymbol{x}} = \dot{J}\dot{\boldsymbol{q}} + J\ddot{\boldsymbol{q}}. \tag{5.13}$$

The equations of motion for the arm can be written as:

$$A(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{b}(\dot{\boldsymbol{q}}, \boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}) + J^T\boldsymbol{\eta}^{int} = \boldsymbol{\tau} \tag{5.14}$$

where $\boldsymbol{\tau}$, A, $\boldsymbol{b}$, $\boldsymbol{g}$, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are from the definition associated to Eq. (5.1). J^T is the transpose of the Jacobian matrix and maps $\boldsymbol{\eta}^{int}$ to equivalent joint torques of the arm. Although we have written the equations of motion for the arm and payload separately, their behaviour is coupled by Eqs. (5.11), (5.12), and (5.13), and to the interaction forces $\boldsymbol{\eta}^{int}$.

We now consider the optimal control problem as before. The equations of motion for the payload become parameterized by s and the vector of interaction forces become,

$$\boldsymbol{\eta}^{int} = \boldsymbol{\lambda}_{obj}\ddot{s} + \boldsymbol{\pi}'_{obj}\dot{s}^2 + \boldsymbol{p}_{obj} \tag{5.15}$$

$$\boldsymbol{\lambda}_{obj} = \Lambda_{obj}\frac{d\boldsymbol{x}}{ds}$$

$$\boldsymbol{\pi}'_{obj} = \Lambda_{obj}\frac{d^2\boldsymbol{x}}{ds^2} + \tilde{\boldsymbol{\pi}}_{obj}$$

where $\tilde{\boldsymbol{\pi}}_{obj}$ is the vector $\boldsymbol{\pi}_{obj}$ with the path velocity $\dot{s}^2$ factored out. Similarly, for the robot arm, the vector of actuator torques for following the path becomes,

$$\boldsymbol{\tau} = \boldsymbol{a}\ddot{s} + \boldsymbol{b}'\dot{s}^2 + \boldsymbol{g} + J^T\boldsymbol{\eta}^{int} \tag{5.16}$$

$$\boldsymbol{a} = A\frac{d\boldsymbol{q}}{ds}$$

$$\boldsymbol{b}' = A\frac{d^2\boldsymbol{q}}{ds^2} + \tilde{\boldsymbol{b}}$$

where $\tilde{\boldsymbol{b}}$ is the vector $\boldsymbol{b}$ with the path velocity $\dot{s}^2$ factored out. We tidy up the notation by substituting Eq. (5.15) into (5.16) to get,

$$\boldsymbol{\tau} = \boldsymbol{a}_1 \ddot{s} + \boldsymbol{a}_2 \dot{s}^2 + \boldsymbol{a}_3 \tag{5.17}$$

The problem is a TPBVP with the equality constraints given by Eqs. (5.15) and (5.17), and with bounds on $\boldsymbol{\eta}^{int}$ and $\boldsymbol{\tau}$ given by Eqs. (5.9) and (5.3) respectively. This again reduces to finding the optimal curve in the s - $\dot{s}$ plane that maximizes the area under it while satisfying the boundary constraints. All that remains is to find the maximum acceleration/deceleration and the maximum possible velocity at each point along the prescribed robot path.

Notice that with exception to (5.9), the constraint equations, inequalities and cost functions are all linear in $\dot{s}^2$, $\ddot{s}$, $\boldsymbol{\tau}$, and $\boldsymbol{\eta}^{int}$. That is, we are close to having a linear programming problem for finding the necessary inputs for the switching point algorithm. To get (5.9) into the required form, we remove the absolute values on the f_z and m_z terms in the inequalities and consider the positive and negative cases. Having removed the absolute values, we now have a linear programming problem in terms of finding the maximum $\dot{s}$ and the maximum/minimum $\ddot{s}$. By treating each positive and negative case seperately and then taking the best result, we can find the maximum acceleration/deceleration and the maximum velocity at each point on the robot path. The details of how to actually transform the above problem into a linear programming one is given in Appendix E. Using the Simplex linear programming algorithm, the desired input for the switching point algorithm can be found, resulting in the "no-slip" minimum-time parameterization of the transfer paths involving one robot.

5.3 The Closed-Chain Robot System

When the multiple arms grasp an object they form a closed-chain system with n degrees of freedom and m actuators, with $m > n$. The equations of motion for this system can be expressed as,

$$\boldsymbol{n}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{o}(\dot{\boldsymbol{q}}, \boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}) = M(\boldsymbol{q})\boldsymbol{\tau} \tag{5.18}$$

where $\boldsymbol{n}$ is the $n \times n$ inertia matrix, $\boldsymbol{o}$ is the $n \times 1$ vector of velocity dependent forces, $\boldsymbol{g}$ is the $n \times 1$ vector of the gravity forces, M is the $m \times n$ matrix that represents the coupling of the redundant actuators in the equations of motion, $\boldsymbol{\tau}$ is the $mx1$ vector of actuators torques, and $\boldsymbol{q}, \dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the generalized coordinates, velocities, and accelerations, respectively.

The equations of motion for the closed-chain given in (5.18) can be rewritten in terms of the the path parameter s, yielding this path-following equation for the actuators,

$$M\boldsymbol{\tau} = \boldsymbol{n}\ddot{s} + \boldsymbol{o}'\dot{s}^2 + \boldsymbol{g} \tag{5.19}$$

$$\boldsymbol{n} = N\frac{d\boldsymbol{q}}{ds}$$

$$\boldsymbol{o}' = N\frac{d^2\boldsymbol{q}}{ds^2} + \tilde{\boldsymbol{o}}$$

where again $\dot{s}$ and $\ddot{s}$ are a measure of the velocity and acceleration of the robot along the path, respectively, and $\tilde{\boldsymbol{o}}$ is the vector $\boldsymbol{o}$ with the path velocity $\dot{s}^2$ factored out (it can be shown that each vector element in $\boldsymbol{o}$ has a $\dot{s}^2$ term).

The constraint equations, inequalities and cost function are all linear in $\dot{s}^2$, $\ddot{s}$, and $\boldsymbol{\tau}$, thus we have the same linear programming problem as in basic problem of Section 5.1. Hence, using Simplex and the switching point algorithm we can find the path-constrained-time-optimal motion of a closed-chain robot system [McCarthy and Bobrow, 1992].

Unfortunately, these time-optimal motions may not be feasible trajectories for the real robot controller to track. For example, the specific method of the controller for partitioning the effort to the arms may be inconsistent with the resulting partitioning of the optimization. Consequently, during the execution of the trajectory the controller may oversaturate one or more of the actuators. We tackle this problem by incorporating the controller strategy into the optimization algorithm. For the purpose of illustration, we consider the multi-arm control strategy based on the *operational space* formulation [Khatib, 1988] and the *virtual linkage* model [Williams and Khatib, 1993]. Other methods such as the *object impedance* control strategy [Schneider and Cannon, 1992] are incorporated in a similar manner.

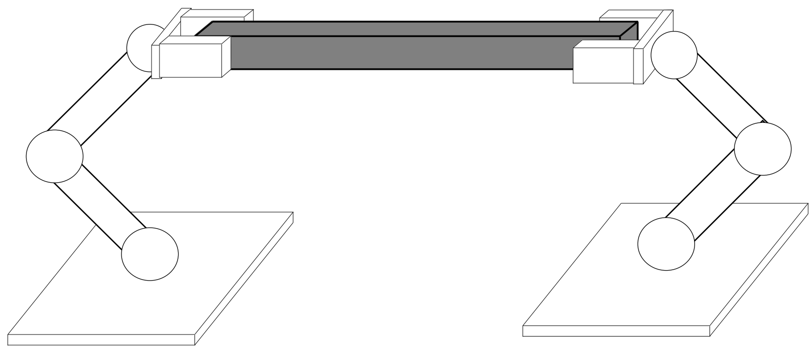


Figure 5.3: Multiple arm manipulation.

5.3.1 Control Consistent Minimum-Time Parameterization

The operational space formulation and the virtual linkage model provide tools to break the control strategy into essentially two components. The bottom level acts to mask the complex non-linear dynamics of the closed-chain system with a simple unit mass rigid body. The top level then controls the behaviour of this simple system, which in turn results in the control of the closed-chain system. We give a brief description of the operational space formulation and the virtual linkage model.

The Operational Space Formulation Consider a multi-arm robot system as shown in Fig. 5.3 with r robots whose end effectors are rigidly connected to the same object. We restrict ourselves to non-redundant manipulators. To the object, a coordinate frame is attached which is called the *operational point*. The position and orientation of this operational point with respect to some inertial reference frame forms a set of generalized coordinates which can be used to describe the configuration for the payload, each robot, or more generally for the closed-chain systems. For simplicity, let the operational point be situated at the mass center of the payload.

Using these generalized coordinates, the equations of motion for the payload can be written as,

$$\Lambda_{obj}(\boldsymbol{x})\ddot{\boldsymbol{x}} + \boldsymbol{p}_{obj}(\boldsymbol{x}) = \boldsymbol{\eta} \tag{5.20}$$

where Λ_{obj} is the $n \times n$ inertia matrix, $\boldsymbol{p}_{obj}$ is the $n \times 1$ vector of gravity dependent terms, $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, and $\ddot{\boldsymbol{x}}$ are the generalized coordinates, velocities, and accelerations respectively, and $\boldsymbol{\eta}$ is the $n \times 1$ vector of generalized operational forces acting on the payload.

Simlarly, the equations of motion for the ith arm can be written as,

$$\Lambda_i(\boldsymbol{x})\ddot{\boldsymbol{x}} + \boldsymbol{\pi}_i(\dot{\boldsymbol{x}}, \boldsymbol{x}) + \boldsymbol{p}_i(\boldsymbol{x}) + \boldsymbol{\eta}_i^{ext} = \boldsymbol{\eta}_i \tag{5.21}$$

where Λ_i is the $n \times n$ inertia matrix, $\boldsymbol{\pi}_i$ is the $n \times 1$ vector of Coriolis and centrifugal forces, $\boldsymbol{p}_i$ is the $n \times 1$ vector of gravity dependent terms, $\boldsymbol{\eta}_i^{ext}$ is the $n \times 1$ vector of external forces, and $\boldsymbol{\eta}_i$ is the $n \times 1$ vector of generalized operational forces. Both $\boldsymbol{\eta}_i$ and $\boldsymbol{\eta}_i^{ext}$ act through the operational point.

We have written separate equations of motion for the payload and each robot arm, but because the robots are rigidly grasping the payload, these equations are coupled by the fact that $\boldsymbol{\eta}$ is the resultant of the forces exerted by each of the r manipulators on the object, that is

$$\boldsymbol{\eta} = \sum_{i=0}^{r} \boldsymbol{\eta}_i^{ext}. \tag{5.22}$$

The complex dynamics of the system is masked with a simple model by selecting the control structure

$$\boldsymbol{\eta} = \hat{\Lambda}_{obj}(\boldsymbol{x})\boldsymbol{\eta}^{\star} + \hat{\boldsymbol{p}}_{obj}(\boldsymbol{x}) \tag{5.23}$$

where $\hat{\Lambda_{obj}}$, and $\hat{\boldsymbol{p}_{obj}}$ represent estimates of Λ_{obj} and $\boldsymbol{p}_{obj}$. With perfect nonlinear decoupling, the payload represented by equation (5.20) under the command (5.23) becomes

$$I_n\ddot{\boldsymbol{x}} = \boldsymbol{\eta}^{\star} \tag{5.24}$$

which is equivalent to a single unit mass I_n moving in the n-dimensional space. We can now design a control law for (5.24) as if it were the open loop dynamics of a system to be controlled [Craig, 1985].

Whatever control structure we devise for computing $\boldsymbol{\eta}^{\star}$, when plugged back into (5.23) we get the net force $\boldsymbol{\eta}$ applied to the payload at the operational point. The next step is to partition $\boldsymbol{\eta}$ to the r arms such that Eq. (5.22) holds. In conjunction with Eq. 5.21 and perfect non-linear decoupling, this determines the generalized active forces for each robot to achieve the desired behaviour of the simple unit mass system.

The Virtual Linkage Model The utility of the virtual linkage model is that it provides a method to partition $\boldsymbol{\eta}$ to the r arms according to Eq. (5.22), while also providing a meaningful characterization of the internal forces and moments in the object. The general idea is to replace the object with an actuated linkage which can resist the same internal forces [Williams, 1994].

Consider a multi-arm system with r robots grasping the same object. A *virtual linkage* is a mechanism connecting the r end effectors with actuated prismatic and spherical joints to characterize the object's internal forces and moments. Since forces applied by the robots act to produce stress throughout the object, by placing $3r - 6$ actuated prismatic joints between pairs of end effectors, the necessary force from the virtual actuators to maintain the rigid grasp yields a characterization of the internal forces. Moreover, since applied moments produce essentially large local stress at the grasp point, by having actuated spherical joints at each grasp point, the necessary torque from the virtual actuator to maintain the rigid grasp yields a characterization of the internal moments. For the two robot system grasping a bar (Fig. 5.3), the corresponding virtual linkage is shown in Fig. 5.4.

Under this framework, it is possible to map the forces and moments exerted by each manipulator to the resultant forces and moments acting on the object and the set of internal forces and moments as characterized by the virtual linkage. By defining a frame on the object (the operational point) the mapping is

$$\begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{t} \\ \boldsymbol{\nu} \end{bmatrix} = G \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{f}_r \\ \boldsymbol{m}_r \end{bmatrix} \tag{5.25}$$

where $\boldsymbol{f}_i$ and $\boldsymbol{m}_i$ are the forces and moments exerted by the i^{th} manipulator on the object described at the operational point, $\boldsymbol{t}$ and $\boldsymbol{\nu}$ are the internal forces and moments

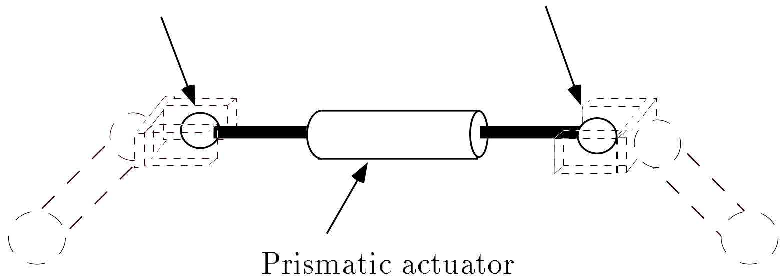


Figure 5.4: Virtual linkage.

respectively, and $\boldsymbol{\eta}$ is the generalized operational force on the payload. The matrix G is called the *grasp description matrix* and is specific to a particular grasp configuration (the location of the grasp points on the object). By using the inverse of G, we get the relationship,

$$\begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{f}_r \\ \boldsymbol{m}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{\eta}_1^{ext} \\ \vdots \\ \boldsymbol{\eta}_r^{ext} \end{bmatrix} = G^{-1} \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{t} \\ \boldsymbol{\nu} \end{bmatrix}. \tag{5.26}$$

Thus, given $\boldsymbol{\eta}$ and by specifying the desired internal forces and moments in the object we get using Eq. (5.26) the corresponding interaction forces $\boldsymbol{\eta}_i^{ext}$ each robot must exert as described at the operational point.

Trajectory Following and Optimization We now combine the results of the operational space formulation and the virtual linkage model, and consider the trajectory following problem. We introduce some idealizations to obtain a direct relationship between trajectory following and the neccessary actuator efforts. Finally, we map this into the required form for the path-constrained-optimal-control problem.

The control structure given by Eq. (5.23) provides the generalized force $\boldsymbol{\eta}$ applied at the operational point of the payload. Using Eq. (5.26) and setting the desired values for internal forces and moments, we get the external forces $\boldsymbol{\eta}_i^{ext}$ felt by each manipulator at the operational point. Finally, the actual torque commands given to the actuator are determined by,

$$\boldsymbol{\tau}_i = J_i^T(\boldsymbol{q}_i)(\Lambda_i(\boldsymbol{x})\boldsymbol{\eta}^\star + \boldsymbol{\pi}_i(\dot{\boldsymbol{x}}, \boldsymbol{x}) + \boldsymbol{p}_i(\boldsymbol{x}) + \boldsymbol{\eta}_i^{ext}) \tag{5.27}$$

where for the i^{th} manipulator J_i is the Jacobian matrix with respect to the operational point, $\boldsymbol{\tau}_i$ is the vector of actuator torques, and $\boldsymbol{q}_i$ is the vector of its joint coordinates. This is assuming a perfectly rigid grasp.

For trajectory following, that is

$$\boldsymbol{x}_d = \boldsymbol{x}(s) \tag{5.28}$$

the servo control law for the simple unit mass body is given by,

$$\boldsymbol{\eta}^\star = \ddot{\boldsymbol{x}}_d + \boldsymbol{k}_v\dot{\boldsymbol{e}} + \boldsymbol{k}_p\boldsymbol{e} \tag{5.29}$$

where $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x}$, $\dot{\boldsymbol{e}} = \dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}$, $\boldsymbol{k}_v$ and $\boldsymbol{k}_p$ are control gains, and $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d$, and $\ddot{\boldsymbol{x}}$ are the desired configurations, velocities, and accelerations respectively that the operational point must track [Craig, 1985]. In the ideal case with zero tracking error and perfect modelling $\boldsymbol{\eta}^\star = \ddot{\boldsymbol{x}}_d$. Thus the net forces and moments at the operational point is

$$\boldsymbol{\eta} = \Lambda_{obj}(\boldsymbol{x}_d)\ddot{\boldsymbol{x}}_d + \boldsymbol{p}_{obj}(\boldsymbol{x}_d). \tag{5.30}$$

This can be rewritten in terms of the path parameter s as,

$$\boldsymbol{\eta} = \boldsymbol{\lambda}_{obj}\ddot{s} + \boldsymbol{\pi}_{obj}\dot{s}^2 + \boldsymbol{p}_{obj} \tag{5.31}$$

$$\boldsymbol{\lambda}_{obj} = \Lambda_{obj}\frac{d\boldsymbol{x}_d}{ds}$$

$$\boldsymbol{\pi}_{obj} = \Lambda_{obj}\frac{d^2\boldsymbol{x}_d}{ds^2}.$$

Similarly, for each robot arm we get,

$$\boldsymbol{\eta}_i = \boldsymbol{\lambda}_i\ddot{s} + \boldsymbol{\pi}_i'\dot{s}^2 + \boldsymbol{p}_i + \boldsymbol{\eta}_i^{ext} \tag{5.32}$$

$$\boldsymbol{\lambda}_i = \Lambda_i \frac{d\boldsymbol{x}_d}{ds}$$

$$\boldsymbol{\pi}'_i = \Lambda_i \frac{d^2\boldsymbol{x}_d}{ds^2} + \tilde{\boldsymbol{\pi}}_i$$

where $\tilde{\boldsymbol{\pi}}_i$ is the vector $\boldsymbol{\pi}_i$ with the path velocity $\dot{s}^2$ factored out.

Now by combining Eq. (5.31), the inverse of the grasp description matrix, and Eq. (5.32) we can determine the actuator efforts of each manipulator for trajectory tracking. We assume the ideal case of a rigid grasp. To minimize the danger of breaking the object or having the grippers slip due to excessive internal forces and moments, we set $\boldsymbol{t}$ and $\boldsymbol{\nu}$ to zero. The resulting equation is,

$$\begin{bmatrix} \boldsymbol{\tau}_1 \\ \vdots \\ \boldsymbol{\tau}_r \end{bmatrix} = \begin{bmatrix} J_1^T & & 0 \\ & \ddots & \\ 0 & & J_r^T \end{bmatrix} \left[\begin{bmatrix} \boldsymbol{\lambda}_1 \ddot{s} + \boldsymbol{\pi}'_1 \dot{s}^2 + \boldsymbol{p}_1 \\ \vdots \\ \boldsymbol{\lambda}_r \ddot{s} + \boldsymbol{\pi}'_r \dot{s}^2 + \boldsymbol{p}_r \end{bmatrix} + G^{-1} \begin{bmatrix} \boldsymbol{\lambda}_{obj} \ddot{s} + \boldsymbol{\pi}_{obj} \dot{s}^2 + \boldsymbol{p}_{obj} \\ 0 \\ 0 \end{bmatrix} \right]. \tag{5.33}$$

This can be rewritten as:

$$\begin{bmatrix} \boldsymbol{\tau}_1 \\ \vdots \\ \boldsymbol{\tau}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_r \end{bmatrix} \ddot{s} + \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_r \end{bmatrix} \dot{s}^2 + \begin{bmatrix} \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_r \end{bmatrix} \tag{5.34}$$

The inequality constraint on the actutator are,

$$\boldsymbol{\tau}^i_{min} \leq \boldsymbol{\tau}_i \leq \boldsymbol{\tau}^i_{max}. \tag{5.35}$$

This has the exact form of the linear programming problem defined in Section 5.1. Hence, using Simplex and the switching point algorithm we can find the "control-consistent" minimum-time parameterization for a transfer path involving multiple arms.

To further impose a "no-slip" constraint the interaction forces $\boldsymbol{\eta}_i^{ext}$ for each robot can be bounded as in Section 5.2. By applying G^{-1} to Eq. (5.31), then transforming the values such that they are described at the grasp point, and after collecting terms we get,

$$\boldsymbol{\eta}_i^{int} = \boldsymbol{d}_{1,i} \ddot{s} + \boldsymbol{d}_{2,i} \dot{s}^2 + \boldsymbol{d}_{3,i}, \tag{5.36}$$

where $\boldsymbol{\eta}_i^{int}$ is $\boldsymbol{\eta}_i^{ext}$ described at the ith grasp point. The time-optimal control problem with constraint Eqs. (5.34) and (5.36) and the corresponding bounds (5.35) and (5.9) has the exact form of the linear programming problem defined in Section 5.2. Hence, using Simplex and the switching point algorithm we can find the "control-consistent-no-slip" minimum-time parameterization for a transfer path involving multiple arms.

5.4 Executing the Time Optimal Robot Motions

We assumed the ideal case when deriving the minimum-time parameterization algorithm. Consequently, for the real robot to actually execute the minimum-time parameterization the dynamic models used by the optimal control algorithms need to be exact, the real robots must perfectly track step responses in the torque (the torque history of the optimal solution may have a bang-bang nature [McCarthy and Bobrow, 1992]), and there can be no tracking errors or disturbances to correct during the motion. Of course in reality, none of these assumptions hold.

However, there are existing experimental robots and robots under development that closely approach this ideal. Torque-controlled manipulators approach the required capability of tracking step responses in the torque history [Pfeffer et al., 1989], [Vischer and Khatib, 1990]. Furthermore, with low level joint torque control, one can neglect to model friction in the joints and drive-train flexibility for the equations of motion of the system [Pfeffer and Cannon, 1993]. This greatly simplifies obtaining accurate dynamic models of the system. In addition, joint torque control allows dynamic decoupling for high-performance control of the fast moving minimum-time trajectories.

By executing the time optimal motions on a high performance robot such as the ARTISAN robot[1] [Khatib and Roth, 1991], the issues of accurate dynamic models and torque control are satisfactorily addressed. Tracking errors however, will always exist. This may result in a saturated actuator being called upon to exert additional torque to correct for tracking errors while following the optimal trajectory. Of course, the robot will fail to correct for the error. Conservative torque bounds must be used

[1]currently under construction at Stanford University

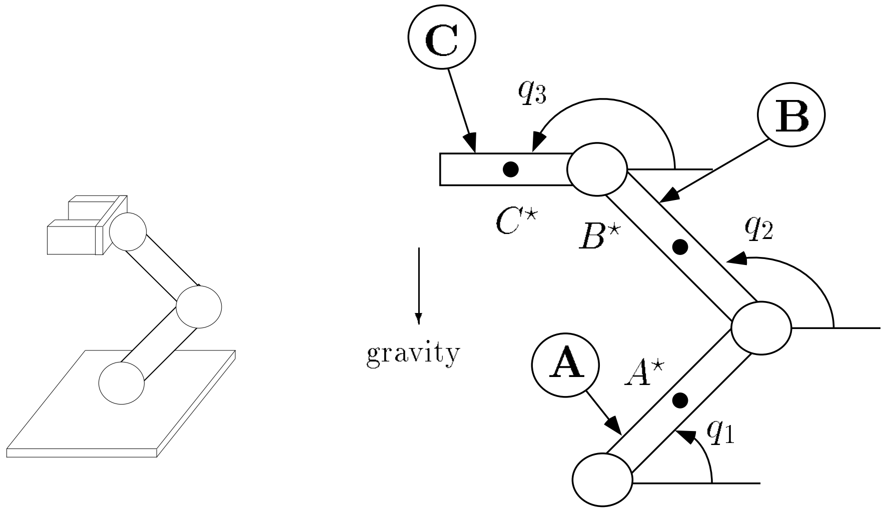


Figure 5.5: The robot and its description.

in the optimization algorithm to ensure reserve torque for dealing with disturbances. The question remains though, how conservative must one be? This is an open problem.

5.5 Simulation Results

We have implemented the path-constrained-time-optimal algorithms for the three cases discussed above.

5.5.1 The Open-Chain Robot

For the open-chain robot we consider the simple planar three degrees of freedom robot shown in Fig. 5.5. The robot consists of three rigid bodies A, B, and C with their mass center at $A^\star$, $B^\star$, and $C^\star$ respectively. The three generalized coordinates are q_1, q_2, and q_3 which measure the rotation of each body with respect to the same reference axis. The robot moves in the vertical plane and has actuators at each of the joints (the shoulder, elbow, and wrist). The parameters of the robot are given in Table 5.1. We use Kane's method [Kane, 1985] to determine the equations of motion.

$l_A = 1.0$m	$l_{A^\star} = 0.5$m	$I_A = 0.2548$ kg-m^2	$m_A = 3.0$ kg	$m_{shoulder} = 5.0$ kg
$l_B = 1.0$m	$l_{B^\star} = 0.5$m	$I_B = 0.2548$ kg-m^2	$m_B = 3.0$ kg	$m_{elbow} = 5.0$ kg
$l_C = 0.4$m	$l_{C^\star} = 0.2$m	$I_C = 0.0299$ kg-m^2	$m_C = 2.0$ kg	$m_{wrist} = 5.0$ kg

Table 5.1: Link parameters.

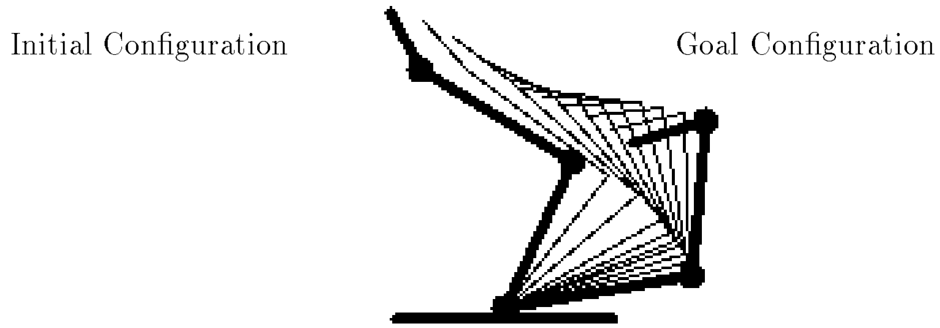


Figure 5.6: Robot path.

For the path shown in Fig. 5.6 a minimum time solution of 0.447s is computed. The torque bounds are,

$$-530.0Nm \leq \tau_{shoulder} \leq 530.0Nm \tag{5.37}$$

$$-300.0Nm \leq \tau_{elbow} \leq 300.0Nm$$

$$-100.0Nm \leq \tau_{wrist} \leq 100.0Nm.$$

The optimal velocity profile is shown in Fig. 5.7 and the torque history is shown in Fig. 5.8. Notice that at all points along the path at least one actuator is saturated.

5.5.2 The Open-Chain Robot Carrying a Payload

We now consider the case where the robot shown in Fig. 5.5 is carrying a payload (see Fig. 5.9). With this particular grasp, the object may potentially slip out of the gripper. The mass of the payload is 1 kg and its moment of inertia is 0.0267 kg·m^2.

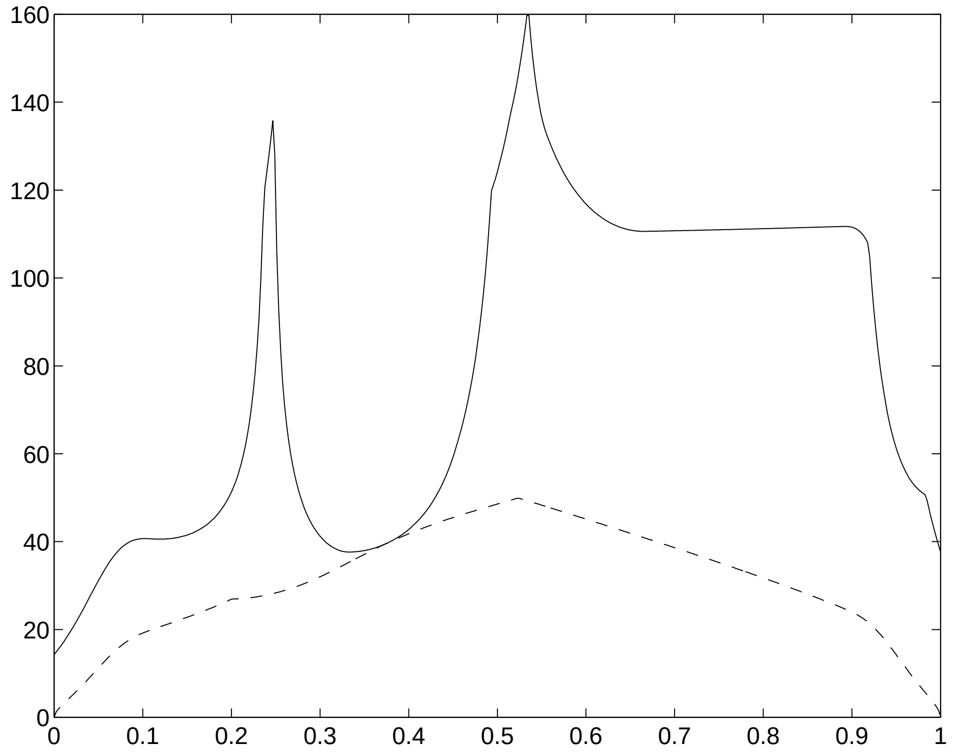


Figure 5.7: Optimal velocity profile for Case 1.

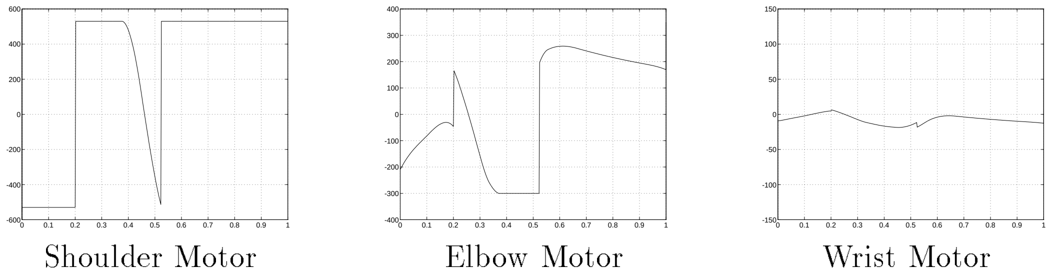


Figure 5.8: Torque history for Case 1.

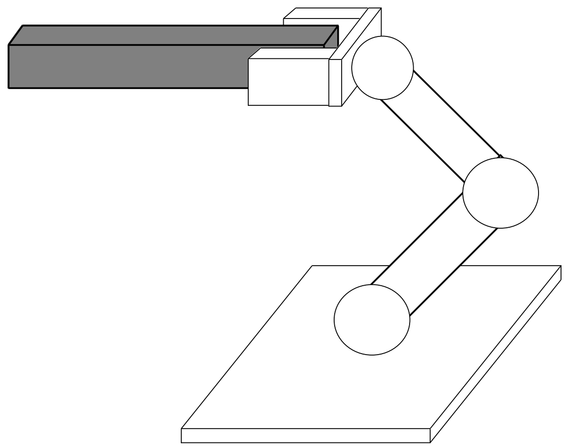


Figure 5.9: The robot and its payload.

The gripping force exerted by each finger is 20N, the coefficient of friction is 0.9, and the α coefficient is 4.2

We first find the minimum-time parameterization for a path without the no-slip constraint. The previous program is used to compute the solution, where the third link of the arm (rigid body C) is the combination of the gripper and the payload. The same specified path for the robot is used (Fig. 5.6) yielding a minimum time solution of 0.473s. The optimal velocity profile is shown in Fig. 5.10 and the corresponding motor torques are shown in Fig. 5.11. The resulting torsion and shear forces between the gripper and the payload along with the friction bounds from Eq. (5.9) are shown in Fig. 5.12. The solid lines are the interaction forces and the dotted lines represent the friction limits. For this optimal solution the friction limits are violated and the payload will slip out of the grasp of the gripper.

The next set of figures is for the minimum-time parameterization with the "no-slip" constraint. The robot path is the same as above. The optimal velocity profile is shown in Fig. 5.13 and the corresponding motor torques are shown in Fig. 5.14. The torsion and shear forces between the gripper and the payload along with the friction bounds from Eq. (5.9) are shown in Fig. 5.15. Again, the solid lines are the interaction forces and the dotted lines represent the friction limits. The minimum

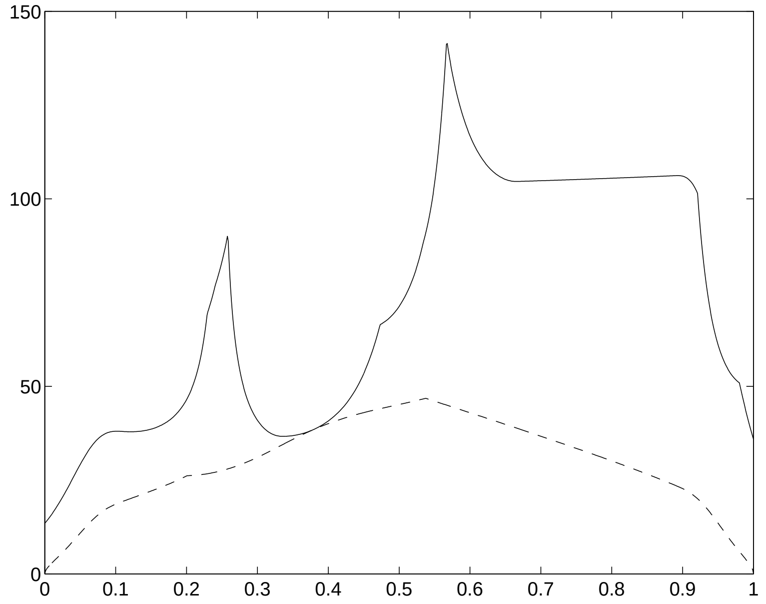


Figure 5.10: Optimal velocity profile for Case 2 (without friction model).

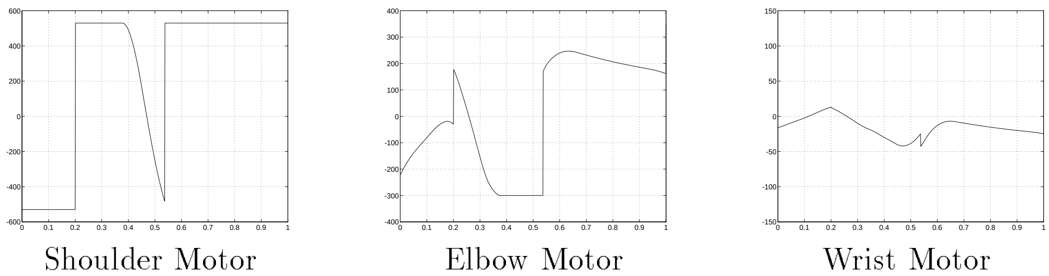


Figure 5.11: Torque history for Case 2 (without friction model).

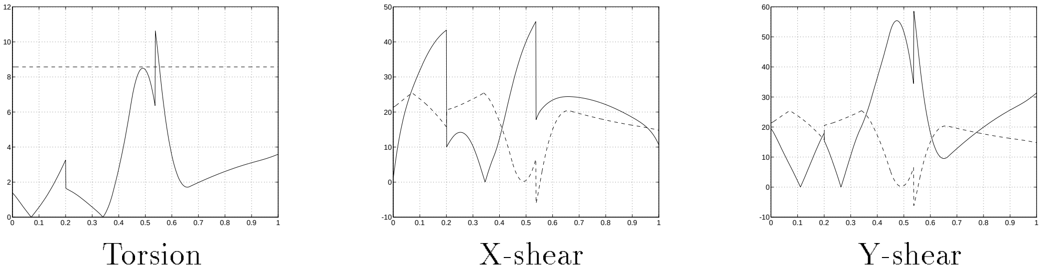


Figure 5.12: Resulting torsion and shear for Case 2 (without friction model).

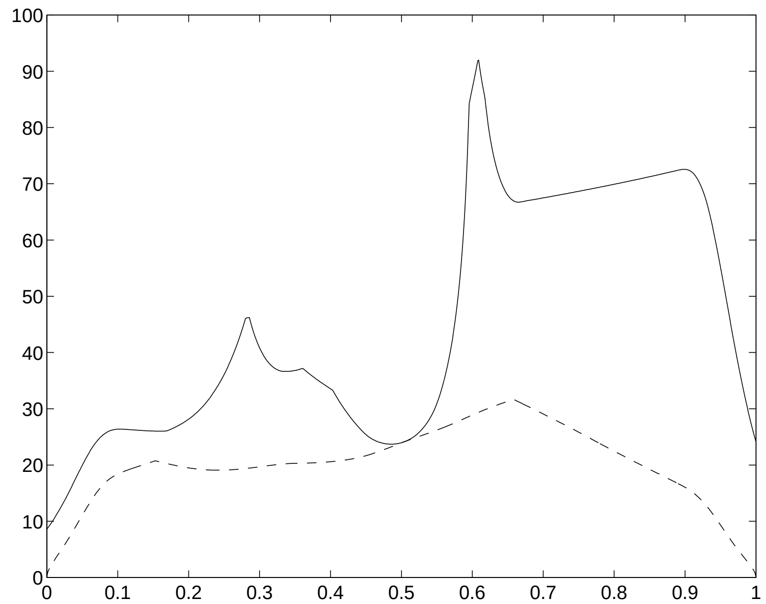
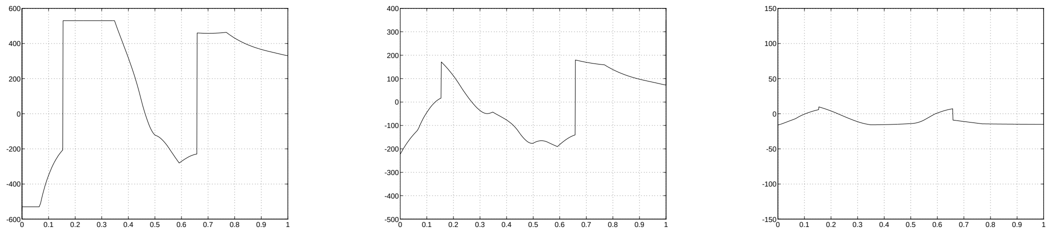
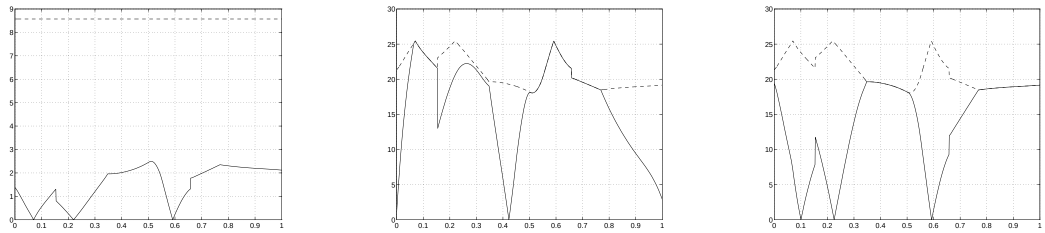


Figure 5.13: Optimal velocity profile for Case 2 (with friction model).



Shoulder Motor | Elbow Motor | Wrist Motor

Figure 5.14: Torque history for Case 2 (with friction model).



Torsion | X-shear | Y-shear

Figure 5.15: Resulting torsion and shear for Case 2 (with friction model).

time solution without the payload slipping is 0.630s.

5.5.3 The Closed-Chain Robot System

We now consider the case where two copies of the previous robot grasp and carry the same payload as shown in Fig. 5.3. The three generalized coordinates of the resulting closed-chain system are the position and orientation of the frame attached to the payload at its center of mass (operational point). The mass of the payload is 3 kg and its moment of inertia is 0.0299 kg$\cdot$m^2. For this simulation we also incorporate the no-slip constraint. In this case, the Howe, Kao, Cutkosky friction model simpifies to the Coulomb friction model, since the interaction moments are zero (internal moments are set to zero in the virtual linkage model). The coefficient of friction is 0.905 and each finger exerts a gripping force of 50 N. The grasp description matrix comes from the example given in [Williams and Khatib, 1993]. For the path shown in Fig. 5.16 a minimum time solution of 0.809s is computed. The torque bounds for each arm are

(a) Initial config.

(b) Motion

Figure 5.16: Closed-chain path.

the same as the previous examples. The optimal velocity profile is shown in Fig. 5.17 and the torque history is shown in Fig. 5.18. The interaction forces at each grasp point is shown in Fig. 5.19, where the solid line is for the X-shear component and the dotted line is for the Y-shear component. The friction limit is ± 32 N for the x and y component of $\boldsymbol{\eta}_i^{int}$, and indeed the object is safe from slipping.

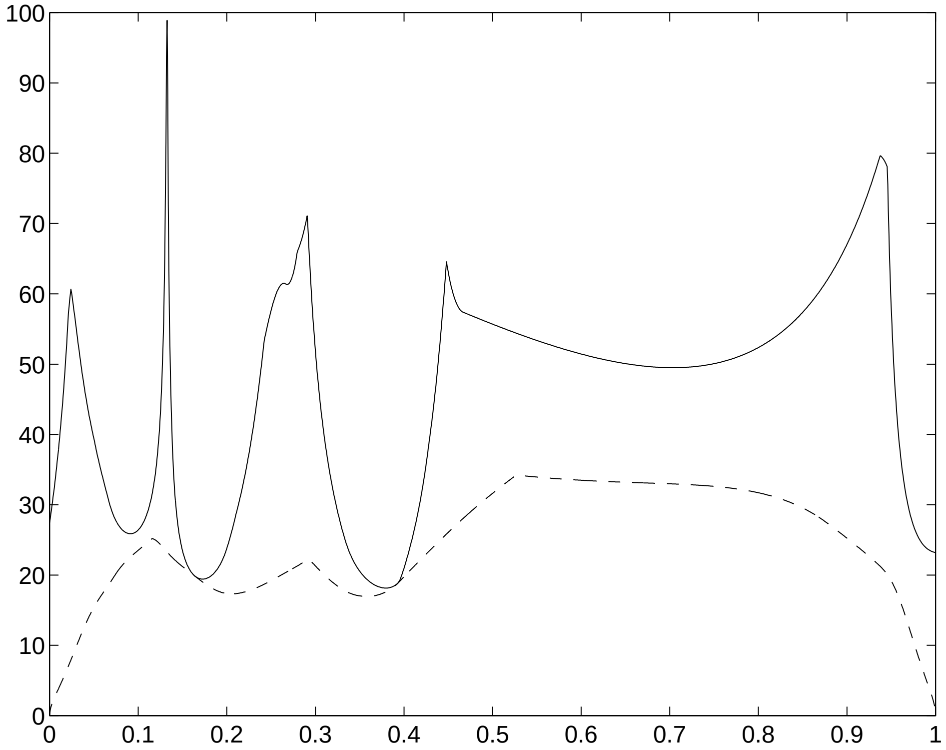


Figure 5.17: Optimal velocity profile for Case 3.

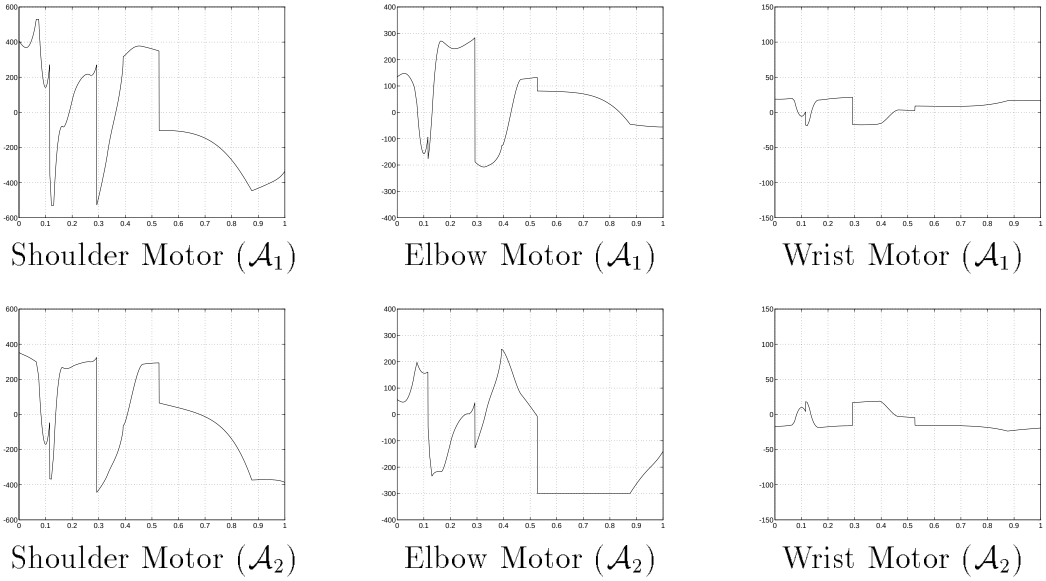


Figure 5.18: Torque history for Case 3.

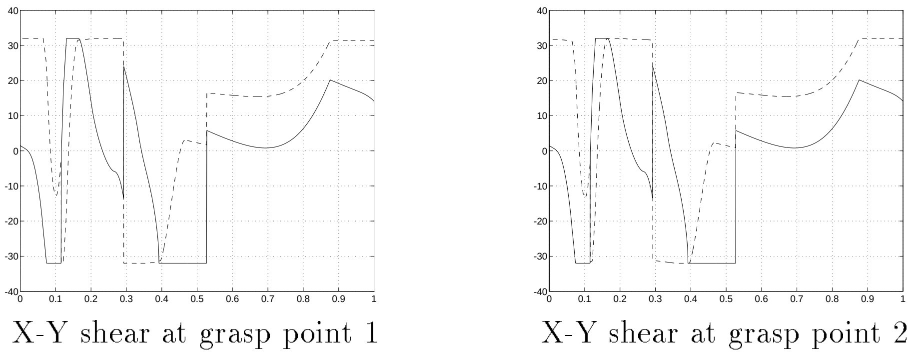


Figure 5.19: Resulting shear forces at the grasp points.

Chapter 6

Conclusion

Previous work in the problem of automatically computing efficient robot trajectories given high-level task commands has shown it to be hard. Theoretical results have their merit, however the development of practical approaches to solving such problems is just as important. To this end, we have presented a practical two-step method for computing multi-arm manipulation trajectories.

For the first step, we have introduced a novel technique for computing the collision-free paths for multiple robot arms to manipulate a movable object to a specified goal location. The approach embeds several simplifications yielding an implemented planner that is not fully general. However, experiments with this planner on various complex examples show that it is quite reliable and efficient in finding manipulation paths when such paths exist. In fact, the planner has succesfully computed rather sophisticated manipulation motions for a system with seventy eight degrees of freedom; an impressive result given that other existing manipulation planners deal with only a few degrees of freedom.

The input to our planner consists of:

- a geometric model of the environment. This includes models of the arms, the movable object and the obstacles;
- a forward and inverse kinematics model for each robot arm;
- a set of static and motion grasp assignments;

- a relatively unobstructive home configuration for each arm;
- the initial and goal location for the movable object and arms.

The output is a series of paths where the multiple arms cooperate to first grasp the object, and then deliver it to the goal location. During the manipulation phase, the arms may ungrasp and regrasp the object for completing the task. The planner takes full advantage of the resources of multiple arms, by utilizing them to execute ungrasp/regrasp operations while suspending the object in the air.

For the second step, we time parameterize the paths found in step one with the minimum-time solution. We do this by utilizing an existing path-constrained-optimal-control algorithm. However, we incorporate additional constraints to deal with the variety of dynamic systems arising in the manipulation motions. Specifically, we add a no-slip constraint to ensure that the payload remains rigidly fixed to the robot, and in the case of multiple arms grasping the same object, we add a control-consistency constraint to ensure that the redundant actuation is dealt with in the same fashion as the multi-arm controller for the real robots.

6.1 Suggestions for Future Work

6.1.1 Improvements to the Manipulation Planner

Two relatively simple improvements to the manipulation planner would be to add a grasp planner to compute feasible grasp points on the object, and a program to take these points and automatically compute the static and motion grasp assignments. For instance, there already exists many grasp planning algorithms that could be integrated with the planner [Pertin-Troccaz, 1989]. To compute the grasp assignments, tools for the design of robots such as the *minimum isotropic acceleration* [Khatib and Burdick, 1987] could be used. The minimum isotropic acceleration for a given arm, is the minimum available, end effector accelerations in all possible directions, with the joint velocities at zero. Thus, a grasp assignment that yields a minimum isotropic acceleration that is at least some ϵ radius for all graspable object configurations (it yields quasi-static motion), would be a motion grasp assignment.

There are some other, more difficult improvements. The planner is currently unable to consider paths where the movable object is placed against some obstacles for regrasping. Future work would involve finding a reliable and efficient heuristic to incorporate this ability into the planner.

Another limitation of the planner is that it only considers a single movable object. However, given the success of RPP in planning the motions of multiple robots, it may be possible to extend RPP* to plan the motions of multiple movable objects. Another possibility may be to extend the method in [Koga et al., 1992] where the sequence of object motions are first determined and then each corresponding manipulation path is found.

6.1.2 Finding Locally Optimal Manipulation Paths

The techniques presented in Chapter 5 provide the basis for further optimization of the manipulation paths. Using variational methods and the minimum-time parameterization algorithm as a cost function, the path shape of the transit and transfer paths could be perturbed in the direction of the negative gradient of the cost function to some locally minimized solution [Bobrow, 1988][Barraquand et al., 1990][Shiller and Dubowsky, 1991]. For a repetitive manipulation motion, a locally time-optimal solution would be highly desirable.

6.1.3 Integrating Trajectory Planning with Real Robots

The two-phase approach to trajectory planning fits directly into the development of a "user-friendly" multi-arm robot system. In [Quinlan and Khatib, 1993], Quinlan and Khatib present the *elastic band* concept as a link between path planning and actual robot execution. They transform the computed path into an elastic band to allow its deformation in real-time. By then utilizing a robot controller with real-time collision avoidance [Khatib, 1986], any unexpected collisons (e.g. due to unexpected obstacles or errors in the control) can be avoided. The manipulation planner of our first step could provide the initial path from which the elastic band would be created.

Currently, Quinlan uses a fast, incremental version of Bobrow's algorithm to time parameterize the elastic bands [Quinlan, 1994]. The time parameterization algorithms in our second step could conceivably be altered into a similar incremental version for time parameterizing the elastic band created from the manipulation path.

By integrating the components from trajectory planning, elastic bands, and motion control, the result is a multi-arm robot system where a person could simply specify the goal location for a movable object and then have the robots automatically execute the motions to complete the task in a robust fashion. The realization of such a system is not so far away.

6.1.4 Dynamic Manipulation Planning

By simultaneously planning for the position, velocity, and acceleration of the system it would be possible to utilize the dynamic effects on the movable object for the purpose of its manipulation [Mason and Lynch, 1993]. For example, in manipulating a heavy object where the object must move in a region where the arms have very little strength, the computed trajectory could be one where the object is accelerated beforehand such that the object essentially coasts through the region of limited arm strength. In fact, this is exactly the sort of strategy humans utilize to "heave" heavy objects into high places. Dynamic manipulation can also be used to regrasp the object. For instance, rather than changing grasps by placing the object on a table, the object could be accelerated in an appropriate fashion such that it slips within the gripper to the desired new grasp. The capabilty to utilize dynamic effects for manipulation would be an exciting addition to the trajectory planner.

6.1.5 Task-Level Animation System

The manipulation planner has applications in computer graphics. The reliability of the planner to compute manipulation motions for many complex tasks makes it suitable as part of an interactive tool to facilitate the animation of scenes.

Initial experiments in integrating the planner with an inverse kinematics algorithm for the human arms show that realistic animation of human manipulation motions can be automatically generated [Koga et al., 1994]. However, in the current implementation, the planner is unable to consider motions where the arms are required to use their redundant degree of freedom to avoid obstacles. For example, a task where the arms must place an object deep into a tight box is almost impossible for our planner. The reason is simply that the inverse kinematics algorithm does not consider obstacle avoidance; it would be useful to devise another algorithm that *does*. Furthermore, incorporating other degrees of freedom of the human such as bending and twisting of the torso would greatly enhance the usefulness of the planner as part of a task-level animation package for human motions.

6.1.6 Ergonomics

Another exciting application of the planner is in ergonomics. By incorporating the same kinematic models of the human arms as in the task-level animation package, realistic motions of the human arms could be simulated to evaluate the design of a product in terms of its usability. This would reduce the number of mock-up models needed to come up with the final design. In addition, by adding human muscle models and considerations of dynamics in the planning, it would be possible to evaluate the design of an assembly process and locate potential dangers to the workers (i.e. muscle strain). This would help to ensure a healthy work forces, and save companies from spending millions of dollars on workers' compensation.

We hope to see continued progress in the development of practical approaches to trajectory planning. Ultimately, this will bring about a drastic simplification in the interaction with robots, in the creation of computer animation, and in the development of useful tools for ergonomics. This dissertation is a step in that direction.

Appendix A

Best-First Planning

We use the best-first planning approach (hereafter called BFP) to find the object path in step one of our 2D manipulation planner as described in Chapter 3. We now give a formal expression of BFP as it applies to finding this object path. For an in-depth explanation of the general BFP algorithm we refer the reader to [Latombe, 1991].

A.1 The BFP* Algorithm

To facilitate the presentation we make use of the following operations dealing with the feasible grasp set:

- GRASP($\boldsymbol{q}_{obj}$) returns the feasible grasp set associated with $\boldsymbol{q}_{obj}$;
- FILTER($\boldsymbol{q}_{obj}$, $\mathcal{FS}$) returns an updated feasible grasp set associated with $\boldsymbol{q}_{obj}$, where the corresponding arm configurations of $R(g, \boldsymbol{q}_{obj})$ for each grasp assignment g in the updated set lie in $\mathcal{CC}_{arms}$;
- `INTERSECT`($\mathcal{FS}$, $\mathcal{FS}'$) returns `true` if $\mathcal{FS}$ and $\mathcal{FS}'$ have a grasp assignment in common and otherwise returns `false`.

BFP uses a list OPEN than contains the leaves of $\mathcal{T}$ sorted by increasing values of the potential function. The following list operations apply to OPEN:

- FIRST(OPEN) removes and then returns the configuration and its associated feasible grasp set in OPEN having the smallest potential value;
- INSERT($\boldsymbol{q}_{obj}$, $\mathcal{FS}$, OPEN) inserts the configuration $\boldsymbol{q}_{obj}$ and its associated feasible grasp set $\mathcal{FS}$ in OPEN;
- `EMPTY`(OPEN) returns `true` if the list OPEN is empty and otherwise returns `false`.

We assume that the potential function $\mathbf{U}(\boldsymbol{q}_{obj})$ is defined such that when $\boldsymbol{q}_{obj}$ is collision-free, the potential is less than some large threshold M, and greater than or equal to M when $\boldsymbol{q}_{obj}$ is in collision.
The formal expression of BFP* is as follows:

procedure BFP*;
begin
    install ($\boldsymbol{q}^i_{obj}$, $\mathcal{FS}^i$) in $\mathcal{T}$; [*initially, $\mathcal{T}$ is the emty tree*]
    INSERT($\boldsymbol{q}^i_{obj}$, $\mathcal{FS}^i$, OPEN); mark $\boldsymbol{q}^i_{obj}$ visited;
    [*initially, all the configurations in $\mathcal{GC}$ are marked "unvisited"*]
    `SUCCESS` $\leftarrow$ `false`;
    while $\neg$`EMPTY`(OPEN) and $\neg$`SUCCESS` do
        begin
            ($\boldsymbol{q}_{obj}, \mathcal{FS}$) $\leftarrow$ FIRST(OPEN);
            for every neighbor $\boldsymbol{q}'_{obj}$ of $\boldsymbol{q}_{obj}$ in $\mathcal{GC}$ do
                if $\mathbf{U}(\boldsymbol{q}'_{obj}) <$ M and $\boldsymbol{q}'_{obj}$ is not visited then
                    begin
                        $\mathcal{FS}' \leftarrow$ FILTER($\boldsymbol{q}'_{obj}$, GRASP($\boldsymbol{q}'_{obj}$));
                        if `INTERSECT`($\mathcal{FS}, \mathcal{FS}'$) then
                            begin
                                install ($\boldsymbol{q}'_{obj}$, $\mathcal{FS}'$) in $\mathcal{T}$ with a pointer
                                towards ($\boldsymbol{q}_{obj}$, $\mathcal{FS}$);
                                INSERT($\boldsymbol{q}'_{obj}$, $\mathcal{FS}'$, OPEN); mark $\boldsymbol{q}'_{obj}$ visited;

if $\boldsymbol{q}'_{obj} = \boldsymbol{q}^g_{obj}$ then `SUCCESS` $\leftarrow$ `true`;
end;
end;
end;
if `SUCCESS` then
return the constructed path by tracing the pointers in $\mathcal{T}$
for $(\boldsymbol{q}^g_{obj}, \mathcal{FS}^g)$ back to $(\boldsymbol{q}^i_{obj}, \mathcal{FS}^i)$;
else return failure;
end;

Procedure BFP* is resolution-complete. This means that for the given discretization of $\mathcal{GC}$, the search will find a collision-free object path if it exists, and otherwise return failure.

A.2 The Potential Function Used in BFP*

We use the following potential function to guide the search described by the above procedure BFP* in the two-dimensional workspace. It is based on the numerical potential function computed by the NF1 procedure described in Chapter 7 of [Latombe, 1991].

NF1 is a general algorithm than can be used to build a potential function in the free subset of $\mathcal{GC}$ that has a global minima at the goal configuration and with only saddle points for local minima. The value of the potential increases as a wave front propagated away from the global minima. Fig. A.1 illustrates the basic idea of how a wave is propagated from the goal configuration. Fig. A.2 is the resulting numerical potential function in $\mathcal{GC}$. The saddle points can be escaped with a slight random perturbation of the configuration, hence with such a potential, a depth-first search can be used to generate a path whenever $\boldsymbol{q}^i_{obj}$ and $\boldsymbol{q}^g_{obj}$ belong to the same connected collision-free subset of $\mathcal{GC}$.

In our implementation, we simplify matters by computing the NF1 potential in a bitmap representation of the workspace for a fixed point on the object called the control point. We denote this bitmap representation of the workspace as $\mathcal{GW}$. Let a be

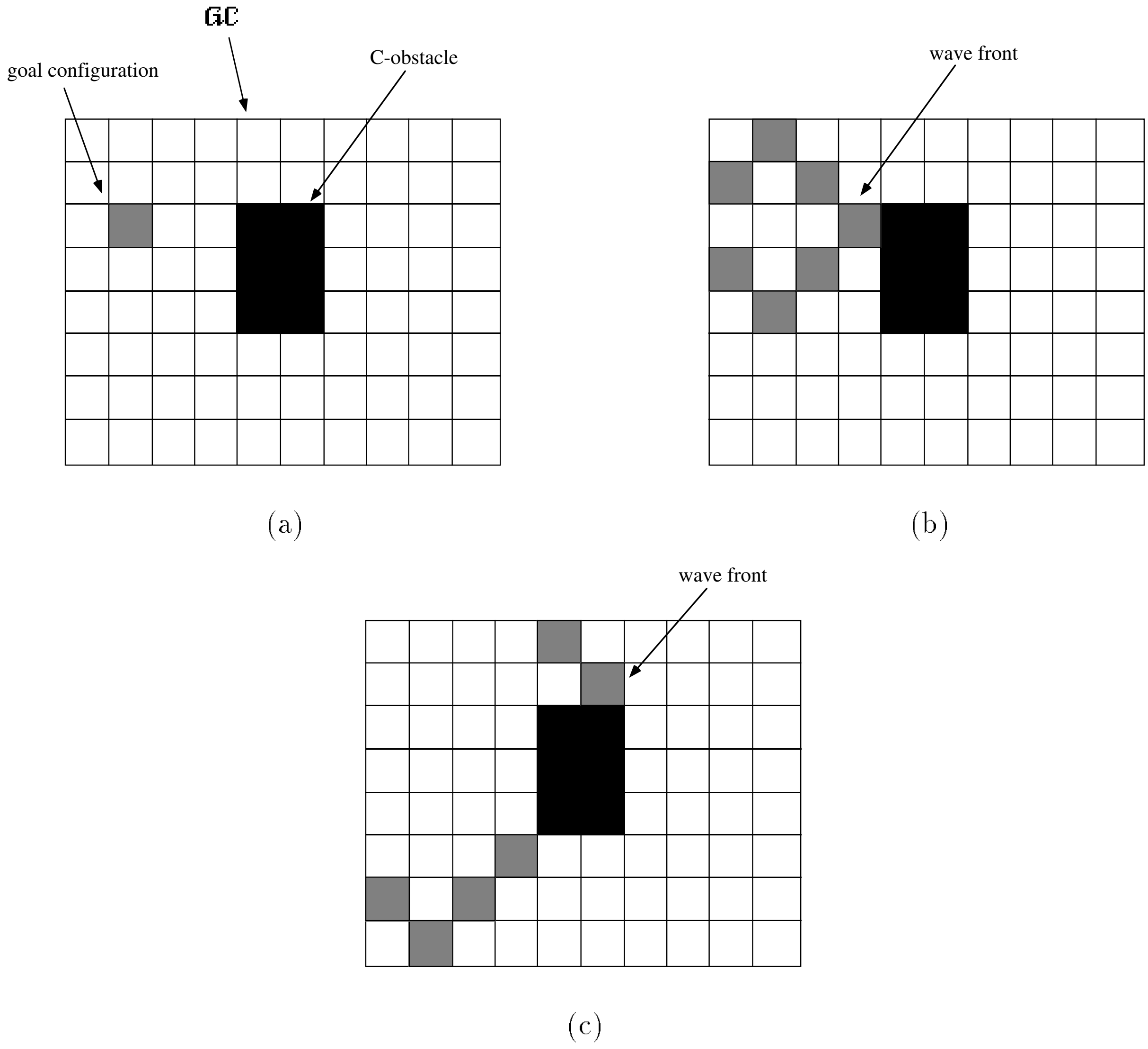


Figure A.1: The wave propogation at various stages of computation.

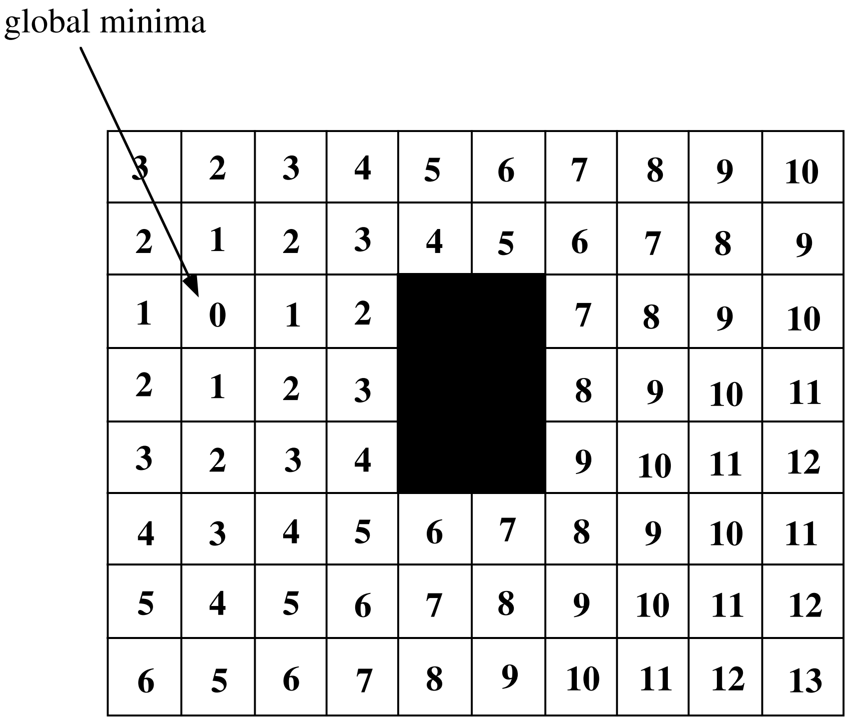


Figure A.2: The numerical potential function NF1.

the control point selected on the object, and let $a(\boldsymbol{q}_{obj})$ denote the (x, y) coordinate of the control point in $\mathcal{GW}$ for an object configuration in $\mathcal{GC}$. The workspace potential $\mathbf{V}$ has the global minima at $a(\boldsymbol{q}^g_{obj})$. The configuration space potential function utilized for finding the object path is then

$$\mathbf{U}(\boldsymbol{q}_{obj}) = \mathbf{V}(\boldsymbol{q}_{obj}) + \rho\Delta(\boldsymbol{q}_{obj}, \boldsymbol{q}^g_{obj}) \tag{A.1}$$

where ρ is a scaling factor and $\Delta(\boldsymbol{q}_{obj}, \boldsymbol{q}^g_{obj})$ returns the absolute value of the minimum rotation needed to move the θ value of $\boldsymbol{q}_{obj}$ to the θ value of $\boldsymbol{q}^g_{obj}$.

Computing $\mathbf{U}$ in this manner saves time in computing the potential function since the NF1 potential is computed in a two-dimensional bitmap instead of a three-dimensional bitmap. The drawback is that $\mathbf{U}$ is a potential function most likely with local minima which are not simple saddle points. BFP however, is very quick to escape such local minima when $\mathcal{GC}$ has dimension three.

We bound $\mathcal{GW}$ within a rectangle and label each point in $\mathcal{GW}$ as "1" if it lies inside an obstacle or along the boundary of $\mathcal{GW}$; it is labelled "0" otherwise. The free-space of $\mathcal{GW}$ denoted as $\mathcal{GW}_{free}$ is the subset of points of $\mathcal{GW}$ that are labelled "0". A formal expression of the NF1 algorithm is as follows:

procedure NF1;
begin
    for every configuration $\boldsymbol{x}$ in $\mathcal{GW}$ do
        begin
            if $\boldsymbol{x} \in \mathcal{GW}_{free}$ then
                $\mathbf{V}(\boldsymbol{x}) \leftarrow$ M;
            else
                $\mathbf{V}(\boldsymbol{x}) \leftarrow$ M+1;
        end;
    $\mathbf{V}(a(\boldsymbol{q}^g_{obj})) \leftarrow 0$; insert $a(\boldsymbol{q}^g_{obj})$ in L_0;
    [L_i, $i = 0, 1, \cdots,$ *is a list of configurations; it is initially empty*]
    for $i = 0, 1, \cdots,$ until L_i is empty do
        for every $\boldsymbol{x}$ in L_i do

for every neighbor $\boldsymbol{x}'$ of $\boldsymbol{x}$ in $\mathcal{GW}_{free}$ do
    if $\mathbf{V}(\boldsymbol{x}') = \mathrm{M}$ then
        begin
            $\mathbf{V}(\boldsymbol{x}') \leftarrow i + h$; [typically h = 1]
            insert $\boldsymbol{x}'$ at the end of L_{i+1};
        end;
end;

Other potential functions could be used such as the NF2 potential described in [Latombe, 1991].

Appendix B

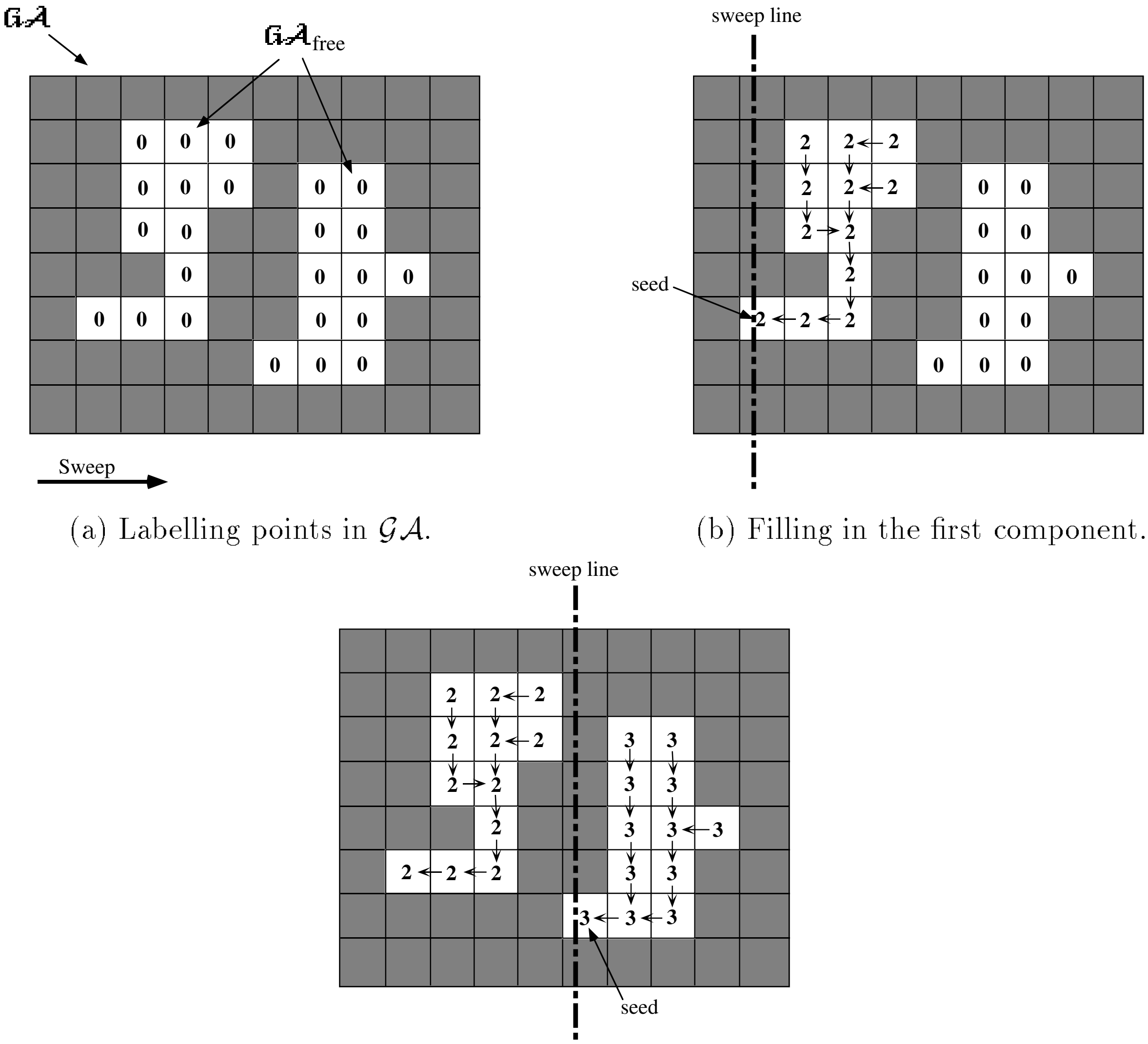
Connected Components of $\mathcal{C}_{arms}$

For finding the object path in our 2D manipulation planner, a method to verify that arm configurations are lying in the same connected component of collision-free arm configurations, namely $\mathcal{CC}_{arms}$, is needed. We make this check using an approximate representation of the freespace of $\mathcal{C}_{arms}$. In this Appendix we give details on how it is constructed.

B.1 Computing the Free Space of $\mathcal{C}_{arms}$

A fine regular grid is thrown over the four-dimensional $\mathcal{C}_{arms}$ which we denote $\mathcal{GA}$. $\mathcal{GA}$ is represented as a parallelepiped and we assume that the joint limits are such that each axis of the parallelepiped is bounded within 0 and 2π radians. For each configuration in $\mathcal{GA}$ we label it with "1" if it is a configuration where the arms collide with the obstacles, each other or violate a joint limit; the configuration is labelled "0" otherwise. The free space of $\mathcal{GA}$, denoted as $\mathcal{GA}_{free}$, is the subset of points in $\mathcal{GA}$ that are labelled "0". Our assumption is that between neighboring points in $\mathcal{GA}_{free}$, there exists a straight line collision-free path connecting them.

B.2 The Connected Components of $\mathcal{GA}$



(a) Labelling points in $\mathcal{GA}$.

(b) Filling in the first component.

(c) Filling in the second component

Figure B.1: The construction of the connected components of $\mathcal{GA}$.

A sweep algorithm groups the discretized collision-free configurations into connected components. During the sweep, for each point encountered in $\mathcal{GA}$ that is labelled "0", the connected component of $\mathcal{GA}_{free}$ in which it belongs is determined. This is done by using the configuration as a "seed" from which a wave is propogated within $\mathcal{GA}_{free}$ until the component is fully explored. A variant of procedure NF1 is used where each configuration visited by the wave propogation is labelled with k, where k is a number greater than "1" and unique to each seed configuration. Then, any two points in $\mathcal{GA}$ that have the same label $k > 1$ are two configurations lying in the same connected component of collision-free arm configurations. Once the component is filled up, the sweep resumes where the next configuration labelled "0" becomes the "seed" for the next component. The steps in constructing this representation is illustrated in Fig.B.1 for a simple two-dimensional $\mathcal{GA}$. The shaded blocks are points in $\mathcal{GA}$ labelled "1". $\mathcal{CC}_{arms}$ is represented in an approximate manner by one of the connected components of $\mathcal{GA}$ computed in this fashion. During the search for the object path, the check whether relevant arm configurations are in $\mathcal{CC}_{arms}$ is simply a lookup operation (i.e., the closest point to the arm configuration in $\mathcal{GA}$ must have the label $k > 1$ associated with $\mathcal{CC}_{arms}$). The worst-case time-complexity to determine the connected components is linear in the number of configurations in the grid $\mathcal{GA}$.

B.3 The Transit Paths and Path Smoothing

To facilitate the search for transit paths within the free space of $\mathcal{GA}$ we encode extra information in each connected component. During the construction of the approximate representation of the components, for each valid neighbor visited by the wave propogation we create a pointer back to its parent configuration. Hence, for any configuration in a component, a path back to the "seed" configuration can be automatically computed by tracing the pointers. Fig. B.1(c) shows an example of the pointers embedded in $\mathcal{GA}$. Consequently, a transit path between two configurations in the same component is constructed by concatenating the two paths that trace back to the seed configuration.

A transit path τ extracted in the aforementioned method needs to be smoothed to produce a more direct path connecting the intial and goal arm configurations. We do so using a simple algorithm that iteratively replaces subpaths of τ of decreasing lengths with straight line segments in the space $\mathbf{R}^4$ containing the grid $\mathcal{GA}$. Each new straight segment is discretized at the resolution of the grid $\mathcal{GA}$ and checked for collision before it is inserted in the new path. The algorithm initially attempts to replace subpaths whose lengths are of the order of the total path length. Then, it considers shorter and shorter subpaths until the resolution of $\mathcal{GA}$ is attained [Latombe, 1991]. More sophisticated techniques may be applied, such as representing the path as an elastic band under tension forces to pull the path tight. Repulsion forces from the obstacles are added to keep the path from "hugging" the obstacles [Quinlan and Khatib, 1993].

Appendix C

Randomized Planning

We use the randomized path planner (hereafter called RPP) to find the object path in step one of our manipulation planner for a three-dimensional workspace. In Chapter 4 we give an overview of RPP as it applies to finding the object path. We now give a formal expression of the algorithm. An in-depth explanation of the general RPP algorithm is given in [Barraquand and Latombe, 1991b] and [Latombe, 1991].

C.1 The RPP* Algorithm

The following is taken from [Latombe, 1991].

We make use of the following definitions to facilitate the presentation:

- τ is a list of configurations representing the path constructed so far;
- LAST(τ) returns the last configuration in the path τ, and it associated feasible grasp set;
- PRODUCT(τ_1, τ_2) returns the list of configurations for the path $\tau_1 \bullet \tau_2$ (the product of τ_1 and τ_2);
- GRADIENT($\boldsymbol{q}_{obj}$, $\mathcal{FS}$) returns the collison-free path generated by following the negative gradient of the potential starting at $\boldsymbol{q}_{obj}$ ($\mathcal{FS}$ is its associated feasible grasp set). The last configuration in the path is at a local minima of

the potential. In addition, within GRADIENT the constraints on the grasp assignment list $\mathcal{GL}$ are satisfied as described in Chapter 4;

- RANDOM-WALK($\boldsymbol{q}_{obj}$, $\mathcal{FS}$, t) returns the path generated by a random walk of duration t starting at $\boldsymbol{q}_{obj}$. The constraints on the grasp list are satisfied as described in Chapter 4;

- RANDOM-TIME returns a random duration;

- BACKTRACK(τ, τ_1, $\cdots$, τ_K) selects a backtracking configuration and returns the path from $\boldsymbol{q}^i_{obj}$ to this configuration; if τ includes a subpath generated by a random motion, then the returned path is a subpath of τ; otherwise it is a subpath of $\tau \bullet \tau_i$, with i randomly chosen between 1 and K;

- `TIME-OUT` returns `true` if a preset time limit is reached; otherwise it returns `false`.

procedure RPP*;
begin
$\tau \leftarrow$ GRADIENT($\boldsymbol{q}^i_{obj}$, $\mathcal{FS}^i$); ($\boldsymbol{q}_{loc}$, $\mathcal{FS}$) $\leftarrow$ LAST(τ);
while $\boldsymbol{q}_{loc} \neq \boldsymbol{q}^g_{obj}$ and $\neg$`TIME-OUT` do
begin
`ESCAPE` $\leftarrow$ `false`;
for $i = 1$ to K until `ESCAPE` do
begin
$t \leftarrow$ RANDOM-TIME;
$\tau_i \leftarrow$ RANDOM-WALK($\boldsymbol{q}_{loc}$, $\mathcal{FS}$, t);
($\boldsymbol{q}_{rand}$, $\mathcal{FS}'$) $\leftarrow$ LAST(τ_i);
$\tau_i \leftarrow$ PRODUCT(τ_i, GRADIENT($\boldsymbol{q}_{rand}$, $\mathcal{FS}'$));
($\boldsymbol{q}'_{loc}$, $\mathcal{FS}'$) $\leftarrow$ LAST(τ_i);
if $\mathbf{U}(\boldsymbol{q}'_{loc}) < \mathbf{U}(\boldsymbol{q}_{loc})$ then
begin

```
                    ESCAPE ← true;
                    τ ← PRODUCT(τ, τ_i);
                end;
            end;
        if ¬ESCAPE then
            begin
                τ ← BACKTRACK(τ, τ_1, ···, τ_K);
                (q_back, FS) ← LAST(τ);
                τ ← PRODUCT(τ, GRADIENT(q_back, FS));
            end;
    end;
end;
```

Without `TIME-OUT`, RPP* is *probabilistically-resolution complete*. This means that if there exist an object path in $\mathcal{GC}$ that satisfies all the requirements (i.e., is collision-free, connects $\boldsymbol{q}^i_{obj}$ to $\boldsymbol{q}^g_{obj}$, and each configuration has the required feasible grasp set), then the probability of finding such a path converges towards 1 when the running time grows towards infinty. Experimental results show that RPP is typically fast to find the path when it exists, hence we can set a time-limit after which with some degree of confidence we can say that no path exists. In our implementation `TIME-OUT` returns `true` when a preset number of backtrack operations are executed.

C.2 The Potential Field Used in RPP*

We use the following potential function to guide the construction of the path during GRADIENT of the above procedure RPP*. It is based on the numerical potential function computed by the NF1 procedure as described in [Latombe, 1991] and in Appendix A. We have added some additional features to the "configuration space potential" which we found from experiments to improve the gradient motion of the object.

We first build a potential field in a bitmap representation of the workspace for fixed points on the object which we call control points. We denote this bitmap representation of the workspace as $\mathcal{GW}$. Let a_1, a_2, and a_3 be the control points selected on the object, and let $a_i(\boldsymbol{q}_{obj})$ denote the (x, y, z) coordinate of the i^{th} control point in $\mathcal{GW}$ for a given object configuration. For each control point i we compute a workspace potential field $\mathbf{V_i}$ using a modified version of procedure NF1 with the global minima at $a_i(\boldsymbol{q}^g_{obj})$. The configuration space potential utilized in GRADIENT is composed of these workspace potentials $\mathbf{V}_i$.

We now describe the modified procedure NF1. The minor change we introduce is to stage the propagation of the wave in selected regions of $\mathcal{GW}$. These regions are based on the space spanned by the workspace of each arm. With N arms, the selected regions are:

- the space spanned by the intersection of all N arm workspaces; all points in this region that are not part of an obstacle are labelled "2";
- the space spanned by the intersection of $N-1$ arm workspaces; all points in this region that are not part of an obstacle are labelled "3";

•

•

•

- the space spanned by only the workspace of a single arm; all points in this region that are not part of an obstacle are labelled "$N+2$";
- the remaining free space labelled "0'; all points in this region are labelled "$N+3$".

In addition, if the point $a_i(\boldsymbol{q}^g_{obj})$ is not labelled "2", then a wave is propagated from $a_i(\boldsymbol{q}^g_{obj})$ where each visited point that is not an obstacle is relabelled "2". This wave is propagated until a point already labelled "2" is encountered.

Prior to calling the modified procedure NF1, we grow the bitmap obstacles in $\mathcal{GW}$ to plug up holes in the workspace in which the object cannot pass through (thus eliminating some of the local minima in the configuration space potential). Procedure NF1 is then invoked where the potential is first computed *only* in the region labelled "2". The neighbour points not labelled "2" are stored in list K_i, where i is the label of the point (i.e., a point labelled "3" is placed in K_3). The variable h in $\mathbf{V}(\boldsymbol{x}') \leftarrow i+h$ is set to 1. When the last L_i becomes empty, the points in K_3 are copied into the list L_{i+1} and computation is resumed. This time however h is incremented by 4 and the potential is computed only in the region labelled "3". This process is repeated until each selected region has been visited by the wave front of NF1. The basic idea is to create a workspace potential for each control point that keeps it within the workspace of the arms, preferably in a region where all arms can reach the object.

A formal expression of the modified NF1 algorithm is as follows:

procedure NF1 (modified);
begin
    for every configuration $\boldsymbol{x}$ in $\mathcal{GW}$ do
        if $\boldsymbol{x} \notin \mathcal{GW}_{free}$ then
            $\mathbf{V}(\boldsymbol{x}) \leftarrow$ M+1;
    $\mathbf{V}(a(\boldsymbol{q}^g_{obj})) \leftarrow 0$; insert $a(\boldsymbol{q}^g_{obj})$ in L_0;
    [L_i, $i = 0, 1, \cdots$, *is a list of configurations; it is initially empty*]
    g = 0; h = 1; P = 2;
    while `REGIONS` do
        begin
            for $i = g, g+1, \cdots$, until L_i is empty do
                for every $\boldsymbol{x}$ in L_i do
                    for every neighbor $\boldsymbol{x}'$ of $\boldsymbol{x}$ in $\mathcal{GW}_{free}$ do
                        begin
                            if $\boldsymbol{x}'$ is labelled P then
                                begin
                                    $\mathbf{V}(\boldsymbol{x}') \leftarrow i + h$;

insert $\boldsymbol{x}'$ at the end of L_{i+1};
end;
else
insert $\boldsymbol{x}'$ at the end of K_j, where j is the label for $\boldsymbol{x}'$;
end;
$g = i$; $P = P + 1$, COPY(K_P, L_{i+1}); h = h+4;
end;
end;

where,

- `REGION` return `true` if there are selected regions still to be visited and `false` otherwise;
- COPY(K_j, L_{i+1}) copies the list of points in K_j into L_{i+1}.

We get the configuration space potential function $\mathbf{U}(\boldsymbol{q}_{obj})$, from the potential values associated to the control points $a_1(\boldsymbol{q}_{obj})$ in $\mathbf{V_1}$, $a_2(\boldsymbol{q}_{obj})$ in $\mathbf{V}_2$, and $a_3(\boldsymbol{q}_{obj})$ in $\mathbf{V_3}$. GRADIENT is slightly modified to accommodate its computation.

For each control point, we fix a frame whose origin is located at a_i. Each frame defines a set of generalized coordinates for the object with respect to some world frame and hence a configuration space. We discretize each space into a fine regular grid and denote them as $\mathcal{GC}_1$, $\mathcal{GC}_2$, and $\mathcal{GC}_3$, where they correspond to the frames attached to a_1, a_2, and a_3, respectively. We use η_i to rewrite configurations in $\mathcal{GC}_i$ as $\boldsymbol{q}_{obj}$, where η_i is the transformation that takes the object configuration defined by the frame attached at a_i and maps it to $\boldsymbol{q}_{obj}$ as defined by some predefined frame on the object. The potential $\mathbf{U}$ is based on one control point at a time, and induces gradient motion in the $\mathcal{GC}_i$ associated to that control point. Consequently, GRADIENT explores not $\mathcal{GC}$ but rather a combination of points in $\mathcal{GC}_1$, $\mathcal{GC}_2$, and $\mathcal{GC}_3$.

Initially, $\mathbf{U}$ is based solely on the value given by a_1. When this no longer yields a motion in GRADIENT, $\mathbf{U}$ is based solely on a_2, and then finally on a_3. The details are as follows.

Let $\mathbf{U}$ be based on control point a_i. A straight line path in $\mathcal{GC}_i$ between the current configuration and the goal is constructed. The line is discretized at the resolution of $\mathcal{GC}_i$ and each point $\boldsymbol{q}$ along the line are given potential values equal to the corresponding value given by $a_i(\boldsymbol{q})$ in $\mathbf{V}_i$. Points not on the line are set to have infinite potential, consequently the negative gradient of the configuration space potential $\mathbf{U}$ is along the line. When GRADIENT no longer yields motion, $\mathbf{U}$ is reset to equal the potential for a_i in $\mathcal{GC}_i$ (i.e., we no longer restrict the motion to the straight line in $\mathcal{GC}_i$). When GRADIENT no longer yields motion then $\mathbf{U}$ is reset to be based on the next control point.

Using $\mathbf{V}_i$ to guide the search in $\mathcal{GC}_i$ only helps to get the control point a_i to its goal coordinate. By switching the focus from a_1 to a_2 and then to a_3, the hope is that in a coarse manner it moves the object to its goal configuration. By initially restricting the motion for each control point along a straight line to the goal configuration, the hope is that in a fine manner it moves the object to achieve $\boldsymbol{q}^g_{obj}$. We found this potential function $\mathbf{U}$ to work well in moving the object through tight openings between obstacles in the workspace.

Appendix D

Linear Programming I

D.1 Deriving the Linear Programming Problem for the Open-Chain Robot

The problem is to find the minimum-time parameterization for the open chain robot. In Section 5.1, we discuss how Bobrow's algorithm solves this problem. In this Appendix we give the details of how the input values to the switching point algorithm are found using a linear programming approach.

The linear programming problem is defined as follows [Press et al., 1988]. Given k independent variables represented as the vector

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}, \tag{D.1}$$

maximize (or minimize) the function

$$z = \boldsymbol{c}^T \boldsymbol{y} \tag{D.2}$$

subject to the primary constraint

$$\boldsymbol{y} \geq \boldsymbol{0} \tag{D.3}$$

and to the additional constraints

$$B_1 \boldsymbol{y} \leq \boldsymbol{d}_1, \tag{D.4}$$

$$B_2\boldsymbol{y} \geq \boldsymbol{d}_2, \tag{D.5}$$

$$B_3\boldsymbol{y} = \boldsymbol{d}_3, \tag{D.6}$$

where B_i is a $m_i \times k$ matrix, $\boldsymbol{d}_i$ is a $m_i \times 1$ vector, and $\boldsymbol{d}_i \geq \boldsymbol{0}$, $(i = 1, 2, 3)$.

We now modify the robot problem to obtain this general form. Let

$$\boldsymbol{x} = \begin{bmatrix} \ddot{s} \\ \dot{s}^2 \\ \boldsymbol{\tau} \end{bmatrix} \tag{D.7}$$

and let it be bounded from above and from below by

$$\boldsymbol{x}_{min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{max} \tag{D.8}$$

where

$$\boldsymbol{x}_{min} = \begin{bmatrix} \ddot{s}_{min} \\ 0 \\ \boldsymbol{\tau}_{min} \end{bmatrix} \tag{D.9}$$

$$\boldsymbol{x}_{max} = \begin{bmatrix} \ddot{s}_{max} \\ \dot{s}^2_{max} \\ \boldsymbol{\tau}_{max} \end{bmatrix}.$$

We introduce $\ddot{s}_{min}$, $\ddot{s}_{max}$, and $\dot{s}^2_{max}$ to simplify the presentation. Because $\ddot{s}$ and $\dot{s}^2$ are to be optimized, we must ensure that $\ddot{s}_{min}$, $\ddot{s}_{max}$, and $\dot{s}^2_{max}$ are bounds way beyond the values obtained during the optimal control of the system.

Using this notation, equation (5.2) can be rewritten as

$$D\boldsymbol{x} = \boldsymbol{h}, \tag{D.10}$$

where

$$D = \begin{bmatrix} -\boldsymbol{a} & -\boldsymbol{b}' & I_{n\times n} \end{bmatrix}, \tag{D.11}$$

$$\boldsymbol{h} = \boldsymbol{g},$$

and I is the $n \times n$ identity matrix.

We now let

$$\boldsymbol{y} = (\boldsymbol{x} - \boldsymbol{x}_{min}) \geq \boldsymbol{0}. \tag{D.12}$$

Furthermore,

$$\boldsymbol{y} \leq (\boldsymbol{x}_{max} - \boldsymbol{x}_{min}). \tag{D.13}$$

Combining (D.10) and (D.12) yields,

$$D\boldsymbol{y} = (\boldsymbol{h} - D\boldsymbol{x}_{min}). \tag{D.14}$$

To make the right hand side of equation (D.14) positive, any negative element and its corresponding row in D is multiplied by -1. We get

$$\tilde{D}\boldsymbol{y} = (\tilde{\boldsymbol{h}} - \tilde{D}\boldsymbol{x}_{min}) \geq \mathbf{0}. \tag{D.15}$$

To find the maximum $\dot{s}$ we set the cost function to be $z = \dot{s}^2 = [1\ 0\ \ldots\ 0]\boldsymbol{y}$. In conjunction with (D.12), (D.13) and (D.15) we have a linear programming problem. We use the Simplex method to find the maximum $\dot{s}$.

To find the maximum (minimum) $\ddot{s}$ for a given $\dot{s} = u$, we make further modifications to the above equations. Since $\dot{s} = u$ (i.e specified) we must extract the $\dot{s}^2$ variable from the $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors accordingly. Consequently, we substitute the D matrix and the $\boldsymbol{h}$ vector with,

$$\tilde{D} = \begin{bmatrix} -\boldsymbol{a} & I_{n\times n} \end{bmatrix}, \tag{D.16}$$

and

$$\tilde{\boldsymbol{h}} = \boldsymbol{g} + \boldsymbol{b}'u^2. \tag{D.17}$$

This is still a linear programming problem and by setting the cost function to $z = \ddot{s} - \ddot{s}_{min} = [1\ 0\ \ldots\ 0]\boldsymbol{y}'$, where $\boldsymbol{y}'$ is the $\boldsymbol{y}$ vector without the $\dot{s}^2$ variable, we get the maximum $\ddot{s}$ for a given $\dot{s}$. To find the minimum $\ddot{s}$ we set $z = -\ddot{s} + \ddot{s}_{min}$.

By utilizing this linear programming approach, the desired input values for the switching point alogorithm can be found, resulting in the minimum-time parameterization of the transit paths.

Appendix E

Linear Programming II

E.1 Deriving the Linear Programming Problem for the Open Chain Robot with a Payload

The problem is to find the minimum-time parameterization for the open chain robot carrying a payload. The payload can not slip out of the grasp of the gripper. In Section 5.2, we discuss how to incorporate a friction model to enforce the no-slip condition. In this Appendix we give the details of how the input values to the switching point algorithm are found using a linear programming approach. The method is an extension of the ideas presented in Appendix D.

Let

$$\boldsymbol{v} = \begin{bmatrix} \ddot{s} \\ \dot{s}^2 \\ \boldsymbol{\tau} \\ \boldsymbol{\eta}^{int} \end{bmatrix} \tag{E.1}$$

and let it be bounded from above and from below by

$$\boldsymbol{v}_{min} \leq \boldsymbol{v} \leq \boldsymbol{v}_{max} \tag{E.2}$$

$$\boldsymbol{v}_{min} = \begin{bmatrix} \ddot{s}_{min} \\ 0 \\ \boldsymbol{\tau}_{min} \\ \boldsymbol{\eta}_{min} \end{bmatrix} \tag{E.3}$$

$$\boldsymbol{v}_{max} = \begin{bmatrix} \ddot{s}_{max} \\ \dot{s}^2_{max} \\ \boldsymbol{\tau}_{max} \\ \boldsymbol{\eta}_{max} \end{bmatrix}$$

We introduce the additional artificial bounds $\boldsymbol{\eta}_{min}$ and $\boldsymbol{\eta}_{max}$ to simplify the presentation. These bounds are way beyond the values set by the constraints (5.9) during the optimal control of the system.

Using this notation, Eqs. (5.17) and (5.15) can be rewritten as

$$Q_1\boldsymbol{v} = \boldsymbol{i}, \tag{E.4}$$

where

$$Q_1 = \begin{bmatrix} -\boldsymbol{a}_1 & -\boldsymbol{a}_2 & I_{n\times n} & 0_{6\times 6} \\ -\boldsymbol{\lambda}_{obj} & -\boldsymbol{\pi}'_{obj} & 0_{n\times n} & I_{6\times 6} \end{bmatrix}, \tag{E.5}$$

$$\boldsymbol{i} = \begin{bmatrix} \boldsymbol{a}_3 \\ \boldsymbol{p}_{obj} \end{bmatrix},$$

and $I_{n\times n}$, $0_{n\times n}$, $I_{6\times 6}$, and $0_{6\times 6}$ are the $n \times n$ and 6×6 identity and zero matrices respectively.

We now let

$$\boldsymbol{y} = (\boldsymbol{v} - \boldsymbol{v}_{min}) \geq \mathbf{0}. \tag{E.6}$$

Furthermore,

$$\boldsymbol{y} \leq (\boldsymbol{v}_{max} - \boldsymbol{v}_{min}). \tag{E.7}$$

Combining (E.4) and (E.6) yields,

$$Q_1\boldsymbol{y} = (\boldsymbol{i} - Q_1\boldsymbol{v}_{min}) = \boldsymbol{r}_1. \tag{E.8}$$

To make the right hand side of the equation positive, any negative element and its corresponding row in Q_1 is multiplied by -1. We get

$$\tilde{Q}_1 \boldsymbol{y} = \tilde{\boldsymbol{r}}_1. \tag{E.9}$$

The bounds on $\boldsymbol{\eta}^{int}$ given by (5.9) can be rewritten in the form of (D.5) and (D.6). We illustrate the steps by converting the upper bound

$$f_x \leq 0.707(\mu(2f_g + |f_z|) - \alpha|m_z|). \tag{E.10}$$

First, we substitute the absolute value on f_z and m_z with the variables $\phi = \pm 1$ and $\psi = \pm 1$. The result is,

$$f_x \leq (1.414\mu f_g + 0.707\mu\phi f_z - 0.707\alpha\psi m_z), \tag{E.11}$$

$$\phi f_z \geq \mathbf{0}, \tag{E.12}$$

$$\psi m_z \geq \mathbf{0}. \tag{E.13}$$

The variables f_x, f_z, and m_z are taken to the left side of the inequality and by changing variables we get

$$\begin{bmatrix} 1 & 0 & -0.707\mu\phi & 0.707\alpha\psi \end{bmatrix} \boldsymbol{\gamma} \leq c, \tag{E.14}$$

$$\begin{bmatrix} 0 & 0 & \phi & 0 \\ 0 & 0 & 0 & \psi \end{bmatrix} \boldsymbol{\gamma} \geq \begin{bmatrix} 0 & 0 & -\phi & 0 \\ 0 & 0 & 0 & -\psi \end{bmatrix} \boldsymbol{\gamma}_{min}, \tag{E.15}$$

where

$$c = (1.414\mu f_g - \begin{bmatrix} 0 & 0 & 0.707\mu\phi & -0.707\alpha\psi \end{bmatrix} \boldsymbol{\gamma}_{min}), \tag{E.16}$$

$$\boldsymbol{\gamma} = \begin{bmatrix} f_x - f_{x,min} \\ f_y - f_{y,min} \\ f_z - f_{z.min} \\ m_z - m_{z,min} \end{bmatrix}, \tag{E.17}$$

and

$$\boldsymbol{\gamma}_{min} = \begin{bmatrix} f_{x,min} \\ f_{y,min} \\ f_{z,min} \\ m_{z,min} \end{bmatrix}. \tag{E.18}$$

The terms $f_{x,min}$, $f_{y,min}$, $f_{z,min}$, and $m_{z,min}$ come from the vector $\boldsymbol{v}_{min}$. (E.14) and (E.15) can be easily rewritten such that $\boldsymbol{\gamma}$ is replaced by $\boldsymbol{y}$.

Following the same derivation we convert (5.9) to

$$Q_2\boldsymbol{\gamma} \geq \boldsymbol{r}_2 \tag{E.19}$$

and

$$Q_3\boldsymbol{\gamma} \leq \boldsymbol{r}_3 \tag{E.20}$$

where

$$Q_2 = \begin{bmatrix} 1 & 0 & 0.707\mu\phi & -0.707\alpha\psi \\ 0 & 1 & 0.707\mu\phi & -0.707\alpha\psi \\ 0 & 0 & \frac{\mu\phi}{\alpha} & \psi \\ 0 & 0 & \phi & 0 \\ 0 & 0 & 0 & \psi \end{bmatrix}, \tag{E.21}$$

$$Q_3 = \begin{bmatrix} 1 & 0 & -0.707\mu\phi & 0.707\alpha\psi \\ 0 & 1 & -0.707\mu\phi & 0.707\alpha\psi \\ 0 & 0 & \frac{-\mu\phi}{\alpha} & \psi \end{bmatrix}, \tag{E.22}$$

$$\boldsymbol{r}_2 = \begin{bmatrix} -1.414\mu f_g \\ -1.414\mu f_g \\ \frac{-2\mu f_g}{\alpha} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & -0.707\mu\phi & 0.707\alpha\psi \\ 0 & 0 & -0.707\mu\phi & 0.707\alpha\psi \\ 0 & 0 & \frac{\mu\phi}{\alpha} & \psi \\ 0 & 0 & -\phi & 0 \\ 0 & 0 & 0 & -\psi \end{bmatrix} \boldsymbol{\gamma}_{min}, \tag{E.23}$$

$$\boldsymbol{r}_3 = \begin{bmatrix} 1.414\mu f_g \\ 1.414\mu f_g \\ \frac{2\mu f_g}{\alpha} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0.707\mu\phi & -0.707\alpha\psi \\ 0 & 0 & 0.707\mu\phi & -0.707\alpha\psi \\ 0 & 0 & \frac{-\mu\phi}{\alpha} & \psi \end{bmatrix} \boldsymbol{\gamma}_{min}. \tag{E.24}$$

To make the right hand side of (E.19) and (E.20) positive, any negative element and its corresponding row in matrix Q_i is multiplied by -1. In addition, this element and its corresponding row in the Q_i matrix are moved to the other inequality. The result is

$$\tilde{Q}_2\boldsymbol{\gamma} \geq \tilde{\boldsymbol{r}}_2 \tag{E.25}$$

and

$$\tilde{Q}_3 \boldsymbol{\gamma} \leq \tilde{\boldsymbol{r}}_3. \tag{E.26}$$

The inequalities (E.25) and (E.26) can be easily rewritten such that $\boldsymbol{\gamma}$ is replaced by $\boldsymbol{y}$. This is done by appropriately embedding the terms of $\tilde{Q}_2$ and $\tilde{Q}_3$ in a zero matrix, and the terms of $\tilde{\boldsymbol{r}}_2$ and $\tilde{\boldsymbol{r}}_3$ in a zero vector. The result is,

$$\bar{Q}_2 \boldsymbol{y} \geq \bar{\boldsymbol{r}}_2 \tag{E.27}$$

and

$$\bar{Q}_3 \boldsymbol{y} \leq \bar{\boldsymbol{r}}_3. \tag{E.28}$$

The constraints (E.6), (E.7), (E.9), (E.27), and (E.28) in conjunction with the cost functions given in Appendix D yield a linear programming problem. We then consider the four linear programming problems with ($\phi = 1, \psi = 1$), ($\phi = 1, \psi = -1$), ($\phi = -1, \psi = 1$), and ($\phi = -1, \psi = -1$) to find the optimal $\dot{s}$ and $\ddot{s}$. The best of the four results is the maximum $\dot{s}$ and the maximum/minimum $\ddot{s}$ we seek.

Bibliography

[Aho, Hopcroft, and Ullman, 1983] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[Alami et al., 1990] R. Alami, T. Siméon and J.P. Laumond, "A Geometrical Approach to Planning Manipulation Tasks: The Case of Discrete Placements and Grasps," *Robotics Research 5*, H. Miura and S. Arimoto, eds., MIT Press, Cambridge, 1990, pp. 453-459.

[Barraquand et al., 1990] J. Barraquand, B. Langlois and J.C. Latombe, "Robot Motion Planning with Many Degrees of Freedom and Dynamic Constraints," *Robotics Research 5*, H. Miura and S. Arimoto, MIT Press, Cambridge, 1990, pp. 435-444.

[Barraquand and Latombe, 1991a] J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," Rep. STAN-CS-89-1257, Department of Computer Science, Stanford University, CA, 1989.

[Barraquand and Latombe, 1991b] J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *The International Journal of Robotics Research*, 10(6), 1991, pp. 628-649.

[Bobrow et al., 1985] J.E. Bobrow, S. Dubowsky, J.S. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, 4(3), 1985, pp. 3-17.

[Bobrow, 1988] J.E. Bobrow, "Optimal Robot Path Planning Using the Minimum-Time Criterion," *IEEE Journal of Robotics and Automation*, 4(4), 1988, pp. 443-449.

[Boneschanscher et al., 1988] N. Boneschanscher, H. Van der Drift, S.J. Buckley, and R.H. Taylor, "Subassembly Stability," *Proceedings of the* 7^{th} *National Conference on Artificial Intelligence*, AAAI 88, 1988, pp. 780-785.

[Canny, 1988] J.F. Canny, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[Canny et al., 1988] J. Canny, B. Donald, J. Reif, and P. Xavier, "On the Complexity of Kinodynamic Planning," *Proceedings of the* 29^{th} *Symposium on the Foundations of Computer Science*, White Plains, NY, 1988, pp. 306-316.

[Craig, 1985] J.J. Craig, *Introduction to Robotics*, Addison-Wesley Publishing Company, Reading, MA, 1986.

[Donald and Xavier, 1989] B. Donald and P. Xavier, "A Provably Good Approximation Algorithm for Optimal-Time Trajectory Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, 1989, pp. 958-963.

[Ferbach and Barraquand, 1993] P. Ferbach and J. Barraquand, "A Penalty Function Method for Constrained Motion Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, pp. 1235-1242.

[Gupta and Zhu, 1994] K.K. Gupta and X. Zhu, "Practical Global Motion Planning for Many Degrees of Freedom: A Novel Approach within Sequential Framework," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, pp. 2038-2043.

[Howe et al., 1990] R.D. Howe, I. Kao, and M.R. Cutkosky, "The Sliding of Robot Fingers Under Combined Torsion and Shear Loading," *Proceedings of the IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1990, pp. 1258-1263.

[Jacobs et al., 1989] P. Jacobs et al., *Planning Guaranteed Near-Time-Optimal Trajectories for a Manipulator in a Cluttered Workspace*, Rep. ESRC 89-20/RAMP 89-15, University of Berkeley, CA, 1989.

[Kane, 1985] T.R. Kane and D.A. Levinson, *Dynamics: Theory and Application*, McGraw-Hill Publishing Company, New York, NY, 1985.

[Kao and Cutkosky, 1992] I. Kao and M.R. Cutkosky, "Quasistatic Manipulation with Compliance and Sliding," *The International Journal of Robotics Research*, 11(1), 1992, pp. 20-40.

[Kavraki and Latombe, 1994] L. Kavraki and J.C. Latombe, "Randomized Preprocessing of Configuration Space for Fast Path Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, pp. 2138-2145.

[Khan and Roth, 1971] M.E. Khan and B. Roth, "The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains," *Journal of Dynamic Systems, Measurement, and Control*, 93(3), 1971, pp. 164-172.

[Khatib, 1986] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, 5(1), 1986, pp. 90-98.

[Khatib and Burdick, 1987] O. Khatib and J. Burdick, "Optimization of Dynamics in Manipulator Design: The operational Space Formulation," *The International Journal of Robotics and Automation*, 2(2), 1987, pp. 90-98.

[Khatib, 1988] O. Khatib, "Object Manipulation in a Multi-Effector Robot System," *Robotics Research 4*, R. Bolles and B. Roth, eds., MIT Press, Cambridge, MA, 1988, pp. 137-144.

[Khatib and Roth, 1991] O. Khatib and B. Roth, "New Robot Mechanisms for New Robot Capabilities," *IEEE/RSJ International Conference on IROS*, Osaka, Japan, 1991, pp. 44-49.

[Koga and Latombe, 1992] Y. Koga and J.C. Latombe, "Experiments in Dual-Arm Manipulation Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, 1992, pp. 2238-2245.

[Koga et al., 1992] Y. Koga, T. Lastennet, J.C. Latombe, and T.Y. Li "Multi-Arm Manipulation Planning," *Proceedings of the 9th International Symposium on Automation and Robotics in Construction*, Tokyo, June 1992, pp. 281-288.

[Koga et al., 1994] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe, "Planning Motions with Intentions", *Proceedings of SIGGRAPH'94*, Orlando, Florida, July 24-29, 1994. To appear in *Computer Graphics* (1994).

[Kondo, 1994] K. Kondo, *Inverse Kinematics of a Human Arm*, Rep. STAN-CS-TR-94-1508, Department of Computer Science, Stanford University, CA, 1994.

[Kondo, 1991] K. Kondo, "Motion Planning with Six Degrees of Freedom by Multi-Strategic Bi-Directional Heuristic Free-Space Enumeration," *IEEE Transactions on Robotics and Automation*, 7(3), 1991, pp. 267-277.

[Kuffner et al., 1994] J. Kuffner, K. Kondo, Y. Koga, J.C. Latombe, "End Game", video transactions of the electronic theater of SIGGRAPH'94.

[Lacquaniti and Soechting, 1982] F. Lacquaniti and J.F. Soechting, "Coordination of Arm and Wrist Motion During A Reaching Task," *The Journal of Neuroscience,* Vol. 2, No. 2, 1982, pp. 399-408.

[Latombe, 1991] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[Latombe, 1992] J.C. Latombe, "Geometry and Search in Motion Planning," *Annals of Mathematics and Artificial Intelligence*, 8(2-4), 1993, pp. 215-227.

[Laugier and Pertin, 1983] C. Laugier and J. Pertin, "Automatic Grasping: A Case Study in Accessibility Analysis," *Advanced Software in Robotics*, A. Danthine and M. Géradin, eds., North-Holland, New York, 1983, pp. 201-214.

[Laumond and Alami, 1988] J.P. Laumond and R. Alami, *A Geometrical Approach to Planning Manipulation Tasks: The Case of a Circular Robot and a Movable Circular Object Amidst Polygonal Obstacles*, Rep. No. 88314, LAAS, Toulouse, 1989.

[Laumond and Alami, 1989] J.P. Laumond and R. Alami, *A Geometrical Approach to Planning Manipulation Tasks in Robotics*, Rep. No. 89261, LAAS, Toulouse, 1989.

[Lozano-Pérez, 1983] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, 32(2), 1983, pp. 108-120.

[Lozano-Pérez et al., 1987] T. Lozano-Pérez et al., "Handey: A Task-Level Robot System," *Robotics Research 4*, R. Bolles and B. Roth, eds., MIT Press, Cambridge, MA, 1988, pp. 29-36.

[Lynch, 1993] K.M. Lynch, "Planning Pushing Paths," *Proceedings of the JSME International Conference on Advanced Mechatronics*, Tokyo, 1993, pp. 451-456.

[Mason and Lynch, 1993] M. Mason and K. Lynch, "Dynamic Manipulation," *IEEE/RSJ International Conference on IROS*, Yokohama, Japan, 1993, pp. 152-159.

[McCarthy and Bobrow, 1992] J.M. McCarty and J.E. Bobrow, "The Number of Saturated Actuators and Constraint Forces During Time-Optimal Movement of a General Robotic System," *IEEE Transactions on Robotics and Automation*, 8(3), 1991, pp. 407-409.

[McCormick, 1982] E.J. McCormick and M.S. Sanders, *Human Factors in Engineering and Design*, McGraw-Hill Book Company, New York, 1982.

[Meier and Bryson, 1987] E.B. Meier and A.W. Bryson, "An Efficient Algorithm for Time Optimal Control of a Two-Link Manipulator," *AIAA Conference on Guidance and Control*, Monterey, CA, 1987, pp. 204-212.

[O'Donnel and Lozano-Pérez, 1989] P.A. O'Donnell and T. Lozano-Pérez, "Deadlock Free and Collision-Free Coordination of Two Robot Manipulators," *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, 1989, pp. 484-489.

[Overmars and Svestka, 1994] M.H. Overmars and P. Svestka, "A Probabilistic Learning Approach to Motion Planning," *Proceedings of the First Workshop on the Algorithmic Foundations of Robotics (WAFR)*, San Francisco, CA, 1994.

[Pardo et al., 1993] G. Pardo-Castellote, T.Y. Li, Y. Koga, R.H. Cannon, J.C. Latombe, and S.A. Schneider, "Experimental Integration of Planning in a Distributed Control System," *Preprints of the 3rd International Symposium on Experimental Robotics*, Kyoto, October 1993.

[Pfeiffer and Johanni, 1987] J. Pfeiffer and R. Johanni, "A Concept for Manipulation Trajectory Planning," *IEEE Journal of Robotics Automation*, RA-3(3), 1987, pp. 115-123.

[Pfeffer et al., 1989] L.E. Pfeffer, O. Khatib, and J. Hake, "Joint Torque Sensory Feedback in the Control of a PUMA Manipulator," *IEEE Transactions on Robotics and Automation*, 5(4), 1991, pp. 418-425.

[Pfeffer and Cannon, 1993] L.E. Pfeffer and R.H. Cannon, Jr., "Experiments with a Dual-Armed, Cooperative, Flexible-Drivetrain Robot System," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, pp. 601-608.

[Pertin-Troccaz, 1989] J. Pertin-Troccaz, "Grasping: A State of the Art," *The Robotics Review 1*, O. Khatib, J.J. Craig, and T. Lozano-Pérez, eds., MIT Press, Cambridge, MA, 1989, pp. 71-98.

[Pieper and Roth, 1969] D. Pieper and B. Roth, "The Kinematics of Manipulators Under Computer Control," *Proceedings of the Second International Congress on Theory of Machines and Mechanisms*, Vol. 2, Zakopane, Poland, 1969, pp. 159-169.

[Press et al., 1988] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK, 1988.

[Quinlan, 1994] S. Quinlan, PhD. dissertation (in preparation)

[Quinlan and Khatib, 1993] S. Quinlan and O. Khatib, "Elastic Bands: Connecting Path Planning and Control," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, pp. 802-807.

[Reif, 1979] J.H. Reif, "Complexity of the Mover's Problem and Generalizations," *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979, pp. 421-427.

[Schneider and Cannon, 1992] S.A. Schneider and R.H. Cannon, "Object Impedance Control for Cooperative Manipulation: Theory and Experimental Results," *IEEE Transactions on Robotics and Automation*, 8(3), 1992, pp. 383-394.

[Shiller and Dubowsky, 1989] Z. Shiller and S. Dubowsky, "Robot Path Planning with Obstacles, Acturator, Gripper, and Payload Constraints," *The International Journal of Robotics Research*, 8(6), 1989, pp. 3-18.

[Shiller and Lu, 1990] Z. Shiller and H. H. Lu, "Robust Computation of Path Constrained Time Optimal Motions," *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 1990.

[Shiller and Dubowsky, 1991] Z. Shiller and S. Dubowsky, "On Computing the Global Time-Optimal Motions of Robotic Manipulators in the Presence of Obstacle," *IEEE Transactions on Robotics and Automation*, 7(6), Dec. 1991.

[Shin and Mackay, 1985] K.G. Shin and N.D. McKay, "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints," *IEEE Transactions on Automatic Control*, AC-30(6), 1985, pp. 531-541.

[Slattery, 1991] R.A. Slattery, *Optimal Control of Closed Chain Robotic Systems*, Ph.D. Dissertation, Dept. of Aerospace Engineering, Stanford University, 1991.

[Soechting and Flanders, 1989] J.F. Soechting and M. Flanders, "Sensorimotor Representations for Pointing to Targets in Three Dimensional Space," *Journal of Neurophysiology,* 62(2), 1989, pp.582-594.

[Tremblay and Cutkosky, 1993] M.R. Tremblay and M.R. Cutkosky, "Estimating Friction Using Incipient Slip Sensing During a Manipulation Task," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, pp. 429-434.

[Tournassoud et al., 1987] P. Tournassoud, T. Lozano-Pérez, and E. Mazer, "Regrasping," *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, pp. 1924-1928.

[Vischer and Khatib, 1990] D. Vischer and O. Khatib, "Design and Development of Torque-Controlled Joints," *Experimental Robotics I*, V. Hayward and O. Khatib, eds., Springer-Verlag, Berlin, Heideleberg, 1990, pp. 271-286.

[Wilfong, 1988] G. Wilfong, "Motion Planning in the Presence of Movable Obstacles," *Proceedings of the 4th ACM Symposium on Computational Geometry*, 1988, pp. 279-288.

[Williams and Khatib, 1993] D. Williams and O. Khatib, "The Virtual Linkage: A Model for Internal Forces in Multi-Grasp Manipulation," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, pp. 1025-1030.

[Williams, 1994] D. Williams, Personal communication, 1994.