Technical Report 1029

# Generating Compliant Motion of Objects with an Articulated Hand

Stephen L. Chiu

MIT Artificial Intelligence Laboratory

*This blank page was inserted to preserve pagination.*

# Generating Compliant Motion of Objects with an Articulated Hand

by

**Stephen L. Chiu**

B.S.M.E., University of California at Berkeley
(1983)

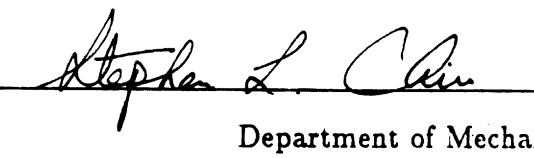
Submitted to the Department of
Mechanical Engineering in
partial fulfillment of the
requirements for the degree of

**Master of Science in Mechanical Engineering**

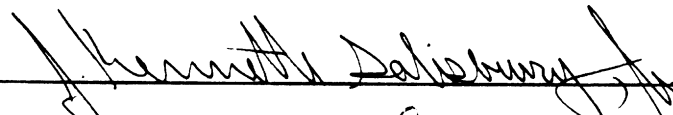
at the

**Massachusetts Institute of Technology**
June 1985

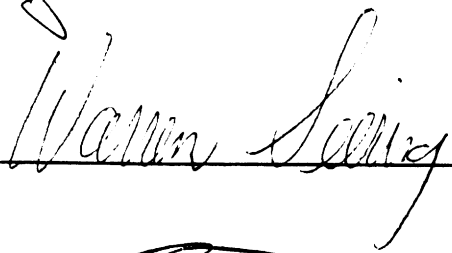
©Massachusetts Institute of Technology, 1985

Signature of Author 

Department of Mechanical Engineering
May 10, 1985

Certified by 

J. Kenneth Salisbury, Jr.
Thesis Supervisor

Certified by 

Warren P. Seering
Thesis Reader

Accepted by

Ain A. Sonin
Chairman, Departmental Graduate Committee

*This empty page was substituted for a blank page in the original document.*

# Generating Compliant Motion of Objects with an Articulated Hand

by

**Stephen L. Chiu**

Submitted to the Department of Mechanical Engineering on May 10, 1985 in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

## Abstract

The flexibility of the robot is the key to its success as a viable aid to production. Flexibility of a robot can be expanded in two directions. The first is to increase the physical generality of the robot such that it can be easily reconfigured to handle a wide variety of tasks. The second direction is to increase the ability of the robot to interact with its environment, such that tasks can still be successfully completed in the presence of uncertainties. The use of articulated hands offers a means for expanding the flexibility of the robot in both directions. Articulated hands are capable of adapting to a wide variety of grasp shapes, hence reducing the need for special tooling. The availability of low mass, high bandwidth joints close to the manipulated object also offers significant improvements in the control of fine motions. This thesis provides a framework for using articulated hands to perform local manipulation of objects. In particular, it addresses the issues in effecting compliant motions of objects in Cartesian space. The Stanford/JPL Hand is used as an example to illustrate a number of concepts. The examples provide an unified methodology for controlling articulated hands grasping with point contacts. We also present a high-level hand programming system based on the methodologies developed in this thesis. Compliant motion of grasped objects and dextrous manipulations can be easily described in the LISP-based hand programming language.

Thesis Supervisor: J. Kenneth Salisbury, Jr.
Research Scientist
MIT Artificial Intelligence Lab

Thesis Reader: Warren P. Seering
Associate Professor of Mechanical Engineering

*This empty page was substituted for a blank page in the original document.*

# Acknowledgments

I would like to thank my advisor, Ken Salisbury, for his guidance and encouragement. The ideas in this thesis are the results of many fruitful discussions with Ken. I would also like to thank Pat O'Donnell for his assistance in the development of the programming system; Pat Sobalvarro for always lending a helpful hand; and all my friends and associates who made the MIT AI Lab such a pleasant place to work.

Financial support for this work was provided by the System Development Foundation and ARPA Contract N00014-82-K-0494.

*This empty page was substituted for a blank page in the original document.*

# Contents


**1 Introduction** **7**
- 1.1 Robotics and Automation . . . . . 7
- 1.2 Applications of Articulated Hands . . . . . 8
- 1.3 Preview . . . . . 8

**2 Manipulation and Contact Constraints** **10**
- 2.1 Introduction . . . . . 10
- 2.2 Kinematic Constraints . . . . . 12
- 2.3 Position Control . . . . . 14
- 2.4 Force Control . . . . . 15

**3 Coordinate Frame Transforms** **17**
- 3.1 Introduction . . . . . 17
- 3.2 Homogeneous Transform Matrix . . . . . 18
- 3.3 Specification of Rotation . . . . . 20
- 3.4 An Example . . . . . 22

**4 Generating Grasp Trajectory** **25**
- 4.1 Introduction . . . . . 25
- 4.2 Hand Kinematic Transforms . . . . . 26
  - 4.2.1 Dealing with Multiple Solutions . . . . . 26
  - 4.2.2 Transforms for the Stanford/JPL Hand . . . . . 29
- 4.3 The Grasp Frame . . . . . 34
  - 4.3.1 Defining a Body-fixed Frame . . . . . 35
  - 4.3.2 Frame Definition for Stanford/JPL Hand . . . . . 36
  - 4.3.3 Using the Grasp Frame . . . . . 39
- 4.4 Generating a Trajectory . . . . . 41
  - 4.4.1 Describing Motion of Objects . . . . . 42
  - 4.4.2 Specifying Motion . . . . . 43
- 4.5 Interpolation in Joint Space . . . . . 47

*This empty page was substituted for a blank page in the original document.*

**5 Grasp Stiffness Control** **50**
5.1 Introduction . . . . . 50
5.2 The Stiffness Matrix . . . . . 52
5.2.1 Diagonal Matrices . . . . . 53
5.2.2 Non-diagonal Matrices . . . . . 54
5.3 Force Application . . . . . 62
5.3.1 Generalized Contact Force . . . . . 63
5.3.2 Generalized Grasp Force . . . . . 64
5.3.3 Grasp Matrix for the Stanford/JPL Hand . . . . . 65
5.3.4 Specifying Force in Alternate Frames . . . . . 68
5.4 Stiffness Control . . . . . 69
5.4.1 The Joint Stiffness Matrix . . . . . 70
5.4.2 Stiffness Control in Alternate Frames . . . . . 71
5.4.3 Object Centered Stiffness Control . . . . . 72
5.5 Impedance Control . . . . . 73

**6 Compliant Motion of Objects** **76**
6.1 Introduction . . . . . 76
6.2 Compliant Trajectory . . . . . 77
6.3 Selecting the Compliance Frame . . . . . 79
6.4 Computational Complexity . . . . . 82

**7 A Hand Control Language** **84**
7.1 Introduction . . . . . 84
7.2 Control Heirarchy and Interaction . . . . . 87
7.3 Basic Functions . . . . . 90
7.3.1 Frame Representation . . . . . 91
7.3.2 Transforming Points and Vectors . . . . . 92
7.3.3 Hand Kinematic Transformations . . . . . 93
7.3.4 Grasp Frame and Grasp Matrix . . . . . 95
7.3.5 Generating a Trajectory . . . . . 96
7.3.6 Sending a Trajectory . . . . . 97
7.3.7 An Example . . . . . 98
7.4 Object-oriented Programming . . . . . 100
7.4.1 Abstract Objects and Message Passing . . . . . 101
7.4.2 The Trajectory Generator Object . . . . . 102
7.4.3 The Parallel Connection Object . . . . . 104
7.5 Constructing a Trajectory . . . . . 105
7.5.1 Motions in Joint Space . . . . . 106
7.5.2 Finger Motion . . . . . 107

*This empty page was substituted for a blank page in the original document.*

7.5.3 Object Motion . . . . . 108
7.5.4 Positioning and Orienting Objects . . . . . 110
7.6 Setting Stiffness . . . . . 111
7.6.1 Joint Stiffness and Force . . . . . 111
7.6.2 Finger Stiffness and Force . . . . . 113
7.6.3 Object Stiffness and Force . . . . . 115
7.7 Sending a Trajectory . . . . . 116
7.8 Programming Examples . . . . . 117
7.8.1 Peg Insertion Using Body-fixed Compliance . . . . . 117
7.8.2 Peg Insertion Using Hand-fixed Compliance . . . . . 118

**8 Conclusions** **121**
8.1 Review . . . . . 121
8.2 The Future . . . . . 123

**A CADR/VAX Messages** **126**

**B The Trajectory Generator** **129**

**C The Parallel Connection** **131**

**References** **132**

*This empty page was substituted for a blank page in the original document.*

# Chapter 1

# Introduction

## 1.1 Robotics and Automation

Over the past twenty years we have seen the evolution of the robot from a novel gadget in research laboratories to a critical component in industrial production. However, the capabilities of the robot are far from that portrayed in science fictions. The applications of current industrial robots are limited to repetitive tasks involving large motions and miminal interactions, such as transferring objects, spray painting, and welding. The robot simply moves through a prescribed sequence of positions in a predictable environment. In this respect, robots are no different from fixed automation. What uniquely distinguishes a robot from fixed automation is its programmability. Through programming, the robot can adapt to different tasks without re-design of its physical configuration. This allows increased flexibility in controlling production, and smaller batches of parts can be manufactured cost effectively.

The flexibility of the robot is the key to its success as a viable aid to production. Flexibility of a robot can be expanded in two directions. The first is to increase the physical generality of the robot such that it can be easily reconfigured to handle a wide variety of tasks. The second direction is to increase the ability of the robot to interact with its environment, such that tasks can still be successfully completed in the presence of uncertainties.

## 1.2 Applications of Articulated Hands

The typical robotic manipulator consists of a six-degrees-of-freedom arm with a simple gripper. Handling parts of different geometries usually requires reconfiguring the gripper by adding special finger shapes. Much of the time in implementing a manipulator system is spent on the design of these special toolings. In tasks requiring handling objects having a variety of geometries, e.g. assembly, the cost involved in the design of these toolings can be considerable. Also, a large portion of the work cycle is spent on tool changing. The development of articulated hands capable of adapting to various grasp shapes seem to offer a solution to this problem.

Close tolerance parts assembly can cause significant forces of interaction between the manipulator and the parts. In the presence of uncertainties in the position and orientation of the parts, a manipulator must be capable of controlling the forces of interaction and allow the geometry of the parts to guide the assembly process. It is difficult to obtain fine control of forces at the gripper from the proximal manipulator joint actuators. Accurate control requires local sensing and exertion of forces. The high bandwidth, powered joints of articulated hands can also be used to provide the necessary local force control.

This work focuses on the application of articulated hands to perform useful manipulation. The increased flexibility in grasping and accurate exertion of forces can significantly extend the capability of an existing manipulator. The low mass links of articulated hands also offer high bandwidth control of motions of objects. This reduces the need to rely on the dynamically complex manipulator for small motions. In effect, an articulated hand can be used as a local high bandwidth manipulator.

## 1.3 Preview

The motivation for this thesis is to provide a famework for using articulated hands to perform local manipulation of objects. In particular, it addresses the

issues in effecting compliant motions of objects in Cartesian space. The Stanford/JPL Hand [Salisbury 1982] is used as an example to illustrate a number of concepts. The examples provide an unified methodology for controlling articulated hands with point contacts.

Before we begin the analysis of hand kinematics and force control, it is important that we have a basic understanding of the task of manipulation. In Chapter 2, we consider the motions of rigid objects as the solutions to a constraint problem. We will examine the task of manipulation in the context of constraint equations.

In Chapter 3, we review the mathematics of coordinate frame transforms which are used extensively throughout this thesis. In Chapter 4, we study the the kinematic transformations for articulated hands. The transformations for the Stanford/JPL Hand are derived as an example. We also derive the *grasp frame*, which provides the necessary link between the position and orientation of a grasped object and the joint coordinates of a hand. The coordinate frame transforms and hand kinematic transforms are combined to describe desired object motions as a set of corresponding joint motions of the hand. Interpolation in joint space during trajectories is also discussed.

In Chapter 5, we study the application of *stiffness control* to articulated hands, where the hand is made to behave as a spring with respect to the grasped object. We also examine the meaning of the *stiffness matrix*, which defines the force/displacement relations. In Chapter 6, we integrate trajectory computation and stiffness control to obtain compliant motion of objects. We also consider some implementational issues.

In Chapter 7, we present a high-level hand programming system based on the concepts and methodologies developed in the thesis. The principal features of the LISP-based hand programming language is described. A programming example for a peg-in-hole insertion task is also given. Finally, in Chapter 8, we review the materials presented in the thesis and give suggestions for future research.

*This empty page was substituted for a blank page in the original document.*

# Chapter 2

# Manipulation and Contact Constraints

## 2.1 Introduction

To describe a point in three dimensional space requires three independent coordinates. To describe a rigid object in three dimensional space requires six independent coordinates, for example, three for locating a reference point on the object and three for specifying the orientation of the object. The number of coordinates that can be independently varied is called the number of *degrees of freedom* of the object. The coordinates which can be indepedently varied are analogous to unknown variables in a mathematical system. To describe the motion of a rigid object is equivalent to specifying sufficient constraint equations on these unknowns such that they are uniquely determined. For an object with $n$ degrees of freedom (DOF), $n$ independent linear constraint equations are required to uniquely describe its motion. If the constraint equations are nonlinear, then there is usually a finite set of possible motions, and additional constraints must be imposed to obtain a unique solution.

As an example, we consider the cube shown in Figure 2.1. When unconstrained, the cube can translate along and rotate about any of the axes. Therefore, six independent coordinates can be varied arbitrarily; hence the uncon-

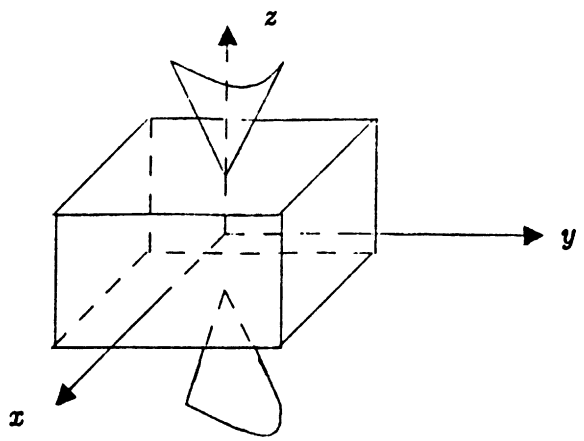



Figure 2.1: Constraining an object reduces the number of DOF

strained cube has six DOF. The correpsonding mathematical system is a system with six unknowns and no constraint equations. When the cube is constrained by the two frictionless point contacts shown, then the cube can no longer translate in the $z$ direction. This fixes the $z$ coordinate of the cube; hence only five coordinates remain which can be instantaneously varied, i.e. the number of DOF is reduced to five. In the corresponding mathematical system, specifying the $z$ coordinate introduces a constraint equation, and the number of unknowns which can be arbitrarily specified reduces to five.

Now suppose the two point contacts are not frictionless, then tranlations in the $x$ and $y$ axes also become constrained.[1] In the corresponding mathematical system, introducing friction at the contacts introduces four additional constraint equations. The number of unknowns which can be arbitrarily specified is reduced to one, the $z$ rotational coordinate. To uniquely determine the configuration, one more constraint is required. The effects of contacts on the DOF of a rigid object have been studied and formalized by Salisbury [1982]. Here we will simply note that the reduction in the DOF of a rigid object depends on the number of

[1] Assuming that the friction is sufficient to resist any translational forces.

contacts and the type of contact, e.g. point contact, line contact, plane contact, with friction, without friction.

From the point of view of manipulation, each DOF represents an arbitrariliness in the possible object motion. To fully *control* the motion of the object, sufficient contacts with the *manipulator* must be made to reduce the DOF of the *object* to zero. Then the possible motions of the object are completely constrained by the motion of the contacts. Hence, manipulation corresponds to using the contacts to impose appropriate constraints such that the possible motions of the object will be uniquely the desired motion. In the following sections we shall study how inconsistent constraints can arise, and how to ensure that the constraints imposed by the manipulator contacts will be consistent.

## 2.2 Kinematic Constraints

A *kinematic constraint* is a constraint imposed by geometry. Violation of a kinematic constraint is a violation of the assumption of rigidity.

When a rigid object is constrained by a set of contacts, its motion must be consistent with those of the contacts, i.e. its motion must satisfy the kinematic constraints. The desired motion of the object is obtained by specifyng appropriate motions of the contacts. Hence, a key issue in manipulation is how to constrain the object and how to generate the desired motions of the constraining contacts.

In terms of the corresponding mathematical system, *the specification of contact motions is equivalent to specification of constraint equations on the possible set of object motions.* Obtaining the desired motion of the object reduces to specifying an appropriate set of constraint equations. This set of constraints must be *consistent with the assumption of rigidity*, otherwise deformation of the object will occur.

As an example, consider the manipulation of a planar object as shown in Figure 2.2. Points $A$ and $B$ are the points of contact between the object and

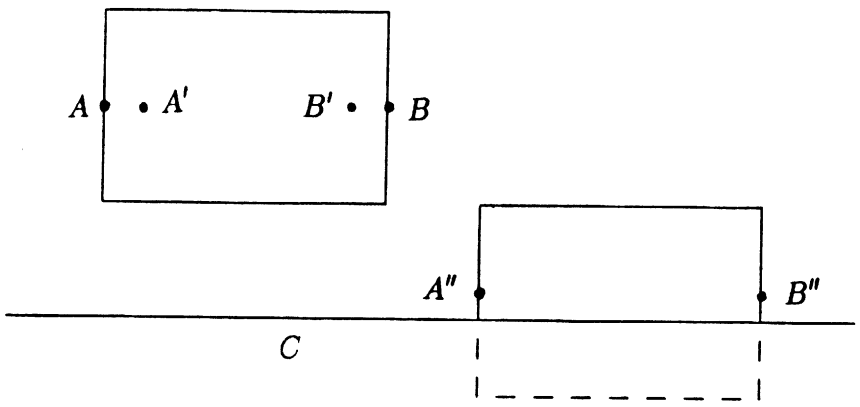



Figure 2.2: Motion of contacts must be consistent with the assumptions of rigidity

the manipulator. Assuming that $A$ and $B$ are rigidly attached to the object, specification of the motion of $A$ and $B$ constitutes a set of constraint equations on the possible motion of the object. The motion of $A$ and $B$ to $A'$ and $B'$ imposes a set of constraint equations which are inconsistent, i.e. there is no solution to the corresponding motion of the rigid objcet. To enforce this set of constraints would violate the assumption of rigidity, and hence cause deformation of the object.

Contacts with the environment also imposes kinematic constraints. The specification of the motion of $A$ and $B$ to $A''$ and $B''$ imposes a set of constraint equations which are inconsistent with that imposed by the surface at $C$. To enforce this set of constriants would require deformation of the object and/or the environment surface.

This example shows that the kinematic constraints imposed by the manipulator contacts must not only be self-consistent, but also be consistent with the constraints imposed by the environment. Self-consistent kinematic constraints can be imposed only if the object shape is precisely known. Kinematic con-

straints consistent with those of the environement can be imposed only if the environment is precisely known.[2]

## 2.3 Position Control

Manipulation can be viewed as the task of using manipulator contacts to impose a set of constraints on the object. The constraints imposed by a contact may be a force constraint or a kinematic constraint. That is, the contact may be used to exert a specified force or to enforce a desired kinematic constraint on the object. Using the manipulator contacts to enforce a desired kinematic constraint is referred to as *position control.*

The desired position and orientation of an object are usually specified in terms of Cartesian coordinates in some fixed coordinate frame. Hence, the required position and orientation of the manipulator contacts are also generally specified in Cartesian coordinates. This requires transforming the specification of the contact motion in Cartesian space into a specification of motion in the manipulator's *natural* coordinate space, e.g. joint angles. This transformation is referred to as the *inverse kinematics* transformation. For the sake of brevity, henceforth the natural coordinate space of the manipulator will be referred to as *joint space.*

The inverse kinematics transformation is generally nonlinear, and hence can yield more than one set of solutions. Additional conditions on the solutions must be specified in order to obtain a unique motion in joint space. The manipulator actuactors are then commanded to execute the desired motion. The issues involved in controlling the joint motion via feedback and dynamics computations will not be discussed here. It will be assumed that the control system can accurately implement the desired motion.

As shown in the previous section, kinematic constraints imposed by the ma-

[2] Free space can be considered as an environment known precisely to impose no kinematic constraints.

nipulator must be self-consistent and consistent with those imposed by the environment. Hence, philosophically, position control can be used only when both the manipulated object and the task environment are precisely known. In practice, the mechanical compliance of the manipulator can absorb the forces generated due to *small* discrepencies in the constraints, hence avoiding deformation of the object.

## 2.4 Force Control

When the object or task environment is not precisely known, using the manipulator contacts as kinematic constraints may lead to inconsistencies. This can also occur during assembly of parts with tolerances less than that of the positional accuracy of the manipulator. Inconsistent kinematic constraints will generate excessive contact interaction forces which may deform the object, the manipulator, or the environment. As the manipulator attempts to enforce the kinematic constraint, excessive actuator torques may also damage the manipulator. Although safeguards can be provided to avoid damages, the task may not be successfully completed.

In order to generate motion of contacts which are kinematically consistent, the manipulator must be able to interact with its environment. In particular it must be able to control the forces of interaction and let the kinematic constraints in its environment *guide* it where appropriate. Using manipulator contacts to impose a desired interaction force is called *force control.*

There are several force control strategies which have been applied to manipulators. Paul and Shimano [1976] proposed a method for controlling forces in a desired direction. In this scheme, the joint which is most closely aligned with the desired force direction is force controlled while the remaining joints are position controlled. Whitney [1977] presented a strategy in which velocity commands are altered based on sensed force. He used an *admittance* matrix to define the desired relation between velocity commands and sensed force. This is in essence a velocity control strategy which has the desired effect of controlling

the contact forces. This strategy can also be conveniently used to move the manipulator end point over obstacles. The problem of specifying manipulator motion to appropriately match the kinematic constraints was studied by Mason [1979]. He proposed that the manipulator motion should be specified to impose a set of *artificial constaints* orthogonal to the *natural constraints* imposed by the environment. Given a constraint surface, the task of the manipulator is then to control the force normal to the surface and velocity tangent to the surface. This approach can be realized in simple tasks by the *hybrid position/force* controller described by Raibert and Craig [1981]. The hybrid scheme involved combining a position and force feedback loop to control position along specified axes and force orthogonal to the axes.

Salisbury [1980] presented a method for controlling the effective *stiffness* of a manipulator. Restoring forces are exerted proportional to the deviation of the endpoint position from a desired nominal trajectory. The manipulator is made stiff in unconstrained directions and compliant in constrained directions. Hogan [1984] argued for this type of strategy from the view point of causal dynamics. He noted that the environment acts as an admittance (i.e. force in, motion out), hence the manipulator should acts as an impedance (i.e. motion in, force out). He proposed a more general strategy which includes stiffness, damping, and inertia terms, i.e the applied forces are functions of the position, velocity, and acceleration. Full control of the apparent *impedance* of the manipulator is not necessary from the stand point of task requirements. Stiffness control is sufficient to control the forces of interaction. However, when dynamic coupling with the environment is considered, stiffness control alone may not be sufficient to maintain stability. Kazerooni [1985] has suggested that the damping terms should be used to ensure stability of the closed-loop system, i.e. the stiffness controller interacting with the environment, and that the inertia terms be used to limit the bandwidth of the controller. The bandwidth of the controller should be chosen to attenuate the effects of high frequency disturbances (e.g. force measurement noise) and unmodeled dynamics.

*This empty page was substituted for a blank page in the original document.*

# Chapter 3

# Coordinate Frame Transforms

## 3.1 Introduction

The task of a manipulator is almost always specified in terms of Cartesian coordinates. For position control, the desired position and orientation of an object is given in Cartesian coordinates. The desired motion is specified as translations along and rotations about Cartesian axes. For force control, the desired forces are specified as forces along and moments about these axes. The transformation of position and force from joint space to Cartesian space is computed with respect to some "absolute" coordinate frame, often one fixed at the base of the manipulator. However, the task is often specified in a coordinate frame different from the "absolute" frame. This requires transforming the desired position or force in the task frame into position or force in the absolute frame.

In this chapter, we will review the mathematics of coordinate frame transforms. There are many methods to represent how one Cartesian coordinate frame is positioned with respect to another. In general, the representation may be divided into two parts: one part describing the relative location of the frame origins and the other describing the relative orientation of the coordinate axes. The relative location of the frame origins is described simply by a vector pointing from one origin to the other. The relative orientation of the coordinate axes, however, have many standard representations. Among the commonly used

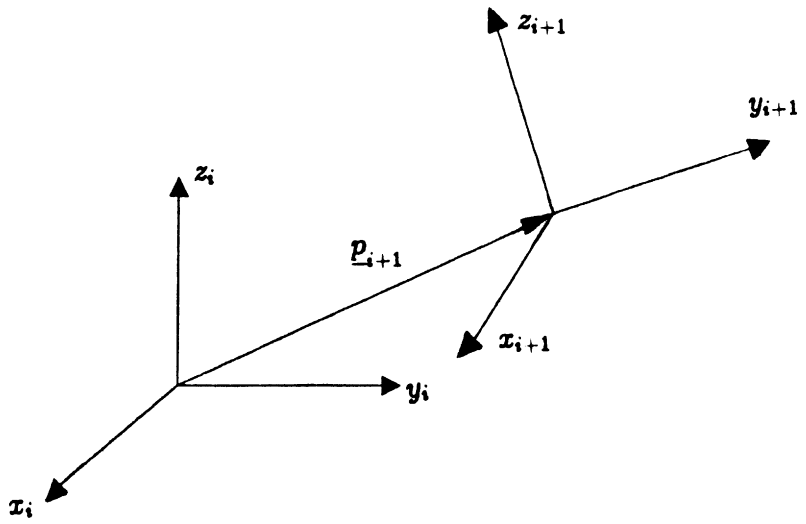



Figure 3.1: Relationships between two coordinate frames

methods are: Euler angles, 3×3 rotation matrices, and quarterions. Here we will study the representation of coordinate frames by 4×4 homogeneous transform matrices, which uses the rotation matrix to describe orientation.

## 3.2 Homogeneous Transform Matrix

The complete representation of location and orientation by a 4×4 homogeneous transform matrix was proposed by Roberts [1965] in connection with computer vision. If the displacement of the frame origin is described by the 3×1 vector $\underline{p}$ and the orientation of the axes by the 3×3 rotation matrix $R$, the homogeneous transform matrix representation is

$$A = \begin{bmatrix} & R & & \underline{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.1}$$

The rotation matrix $R$ is simply a matrix of direction cosines expressing the components of the $x$, $y$, and $z$ axis of one coordinate frame in another. The realtionship between the two frames i and i + 1 in Figure 3.1 is given by

$$A_{i+1} = \begin{bmatrix} x_{i+1x} & y_{i+1x} & z_{i+1x} & p_{i+1x} \\ x_{i+1y} & y_{i+1y} & z_{i+1y} & p_{i+1y} \\ x_{i+1z} & y_{i+1z} & z_{i+1z} & p_{i+1z} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{3.2}$$

The first column is composed of the components in frame i of the $x$ unit vector of frame i + 1. The second and third columns are composed of the components of the $y$ and $z$ unit vectors, respectively. The last column is the vector pointing from the orgin of frame i to the origin of frame i + 1.

The homogeneous representation of a point $(x, y, z)$ is the column vector

$$\underline{x} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{3.3}$$

If a point is located by $\underline{x}_{i+1}$ in frame i + 1, then its location in frame i is

$$\underline{x}_i = A_{i+1}\, \underline{x}_{i+1} \tag{3.4}$$

Let i + 2 be a frame described relative to frame i + 1 by $A_{i+2}$, then

$$\underline{x}_{i+1} = A_{i+2}\, \underline{x}_{i+2}$$

and hence

$$\underline{x}_i = [A_{i+1} A_{i+2}]\, \underline{x}_{i+2} \ .$$

Therefore we see that succesive relative frame transforms can be reduced to one composite transform matrix by simple matrix multiplications. If there are $n$ succesive relative frames, then a point expressed in frame n as $\underline{x}_n$ is expressed in the base frame 0 by

$$\underline{x}_0 = [A_1 A_2 \cdots A_n]\, \underline{x}_n \ .$$

Coordinate transfomation in the opposite direction is accomplished by inversion of the transform matrix. For example, if the location of a point is given in terms of frame i, to find its location in frame i + 1, we use

$$\underline{x}_{i+1} = A_{i+1}^{-1}\, \underline{x}_i . \tag{3.5}$$

Since the inverse of a rotation matrix is its transpose, the inverse of a transform matrix (Equation 3.1) can be shown to be

$$A^{-1} = \begin{bmatrix} & R^T & & -R^T\underline{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

The homogeneous representation of a direction *vector* $(x, y, z)$ is

$$\underline{v} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} . \tag{3.6}$$

Similarly, if a direction vector is expressed as $\underline{v}_{i+1}$ in frame $\mathbf{i+1}$, then in frame $\mathbf{i}$ it is expressed as

$$\underline{v}_i = A_{i+1}\,\underline{v}_{i+1} .$$

Because the last row of $\underline{v}_{i+1}$ is zero, it is easily seen that the displacement of the frame origin has no effect on the result vector (see Equation 3.2). This is expected, since the magnitude of a vector is constant with respect to any coordinate frame, only the direction vary. Hence, transforming a vector reduces to a simple multiplication by the rotation matrix $R$.

In addition to using the transform matrix to represent a coordinate frame, it can be used to represent the position and orientation of a rigid object. That is, the transform matrix can be used to represent a frame *fixed* in a rigid object. The position and orientation of the frame then become attributes of the object.

## 3.3 Specification of Rotation

The relationship between two adjacent frames can also be specified as an equivalent rotation followed by a translation of the frame origin. For the two frames shown in Figure 3.2, we envision that frame $\mathbf{i+1}$ originally coincides with frame $\mathbf{i}$. Frame $\mathbf{i+1}$ is first rotated[1] about the unit vector $\underline{n}$ through the

[1]Positive rotation corresponds to a right hand screw

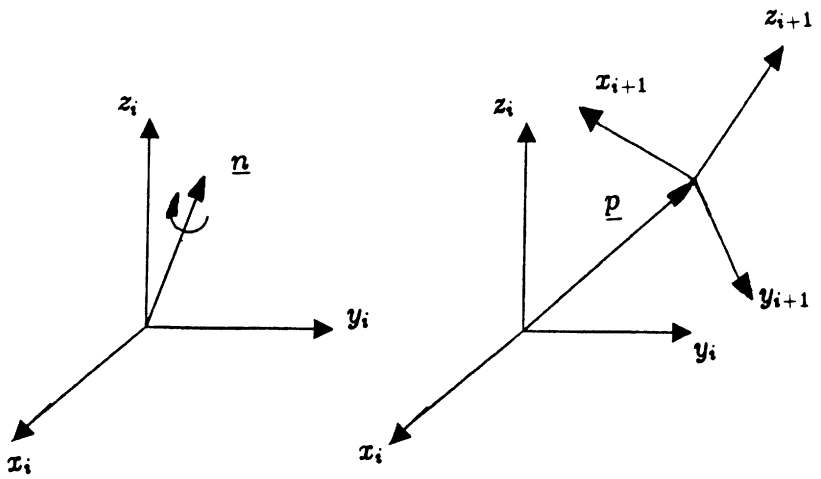



Figure 3.2: Describing relation of frames as a rotation followed by a translation

angle $\theta$. Then the origin of frame $\mathbf{i+1}$ is translated along the vector $\underline{p}$. This representation is physically more meaningful than the transform matrix when describing the position and orientation of an object. Instead of stating *where* an object is, we can now describe *how* it got there. Replacing the 3×3 rotation matrix $R$ by the pair $[\underline{n}, \theta]$ also conserves data storage.

Given a rotation matrix $R$ composed of the column vectors $(\underline{u}, \underline{v}, \underline{w})$

$$R = \left[ \begin{array}{ccc} \underline{u} & \underline{v} & \underline{w} \end{array} \right] \tag{3.7}$$

the correponding pair $[\underline{n}, \theta]$ is found from the formulae

$$\cos\theta = \frac{1}{2}\left(u_x + v_y + w_z - 1\right) \tag{3.8}$$

$$n_x = \operatorname{sgn}\left(v_z - w_y\right)\sqrt{\frac{u_x - \cos\theta}{1 - \cos\theta}} \tag{3.9}$$

$$n_y = \operatorname{sgn}\left(w_x - u_z\right)\sqrt{\frac{v_y - \cos\theta}{1 - \cos\theta}} \tag{3.10}$$

$$n_z = \operatorname{sgn}\left(u_y - v_x\right)\sqrt{\frac{w_z - \cos\theta}{1 - \cos\theta}}. \tag{3.11}$$

When the angle $\theta$ is small, numerical computation of the components of $\underline{n}$ become inaccurate. Paul [1981] presented a method for more accurately determining these values. Whitney [1972] gives an alternate approach in which $\underline{n}$ is found as an eigenvector of the rotation matrix.

The rotation matrix corresponding to a specification of $[\underline{n}, \theta]$ is found from the formula

$$R(\underline{n},\theta) = \begin{bmatrix} n_x n_x \text{vers}\,\theta + \cos\theta & n_y n_x \text{vers}\,\theta - n_z \sin\theta & n_z n_x \text{vers}\,\theta + n_y \sin\theta \\ n_x n_y \text{vers}\,\theta + n_z \sin\theta & n_y n_y \text{vers}\,\theta + \cos\theta & n_z n_y \text{vers}\,\theta - n_x \sin\theta \\ n_x n_z \text{vers}\,\theta - n_y \sin\theta & n_y n_z \text{vers}\,\theta + n_x \sin\theta & n_z n_z \text{vers}\,\theta + \cos\theta \end{bmatrix} \tag{3.12}$$

where

$$\text{vers}\,\theta = \text{versine}\,\theta = 1 - \cos\theta \ .$$

Equation 3.12 is very useful for locating points fixed on objects, as will be shown next.

## 3.4 An Example

Manipulation by position control requires computing the appropriate manipulator contact motion which will enforce a desired object motion. Assuming no slip has occured and that rolling is negligible, a point contact will remain fixed with respect to the object throughout the motion. As a preview of the materials in the next chapter, we will consider an example which involves locating a fixed point on an object after a prescribed object motion.

Consider the rigid cup shown in Figure 3.3. Point $P$ is fixed on the side of the cup and is located by $\underline{x}_o$. in the "absolute" coordinate frame o. Let c be a coordinate frame related to o by the transform matrix $A_c$. If the cup is rotated by $\theta$ about $\hat{x}_c$ and then translated by the distance $h$ along $\hat{z}_c$, where is point $P$ with respect to coordinate frame o ?

First we find the location of $P$ relative to frame c from

$$\underline{x}_c = A_c^{-1}\,\underline{x}_o \ .$$

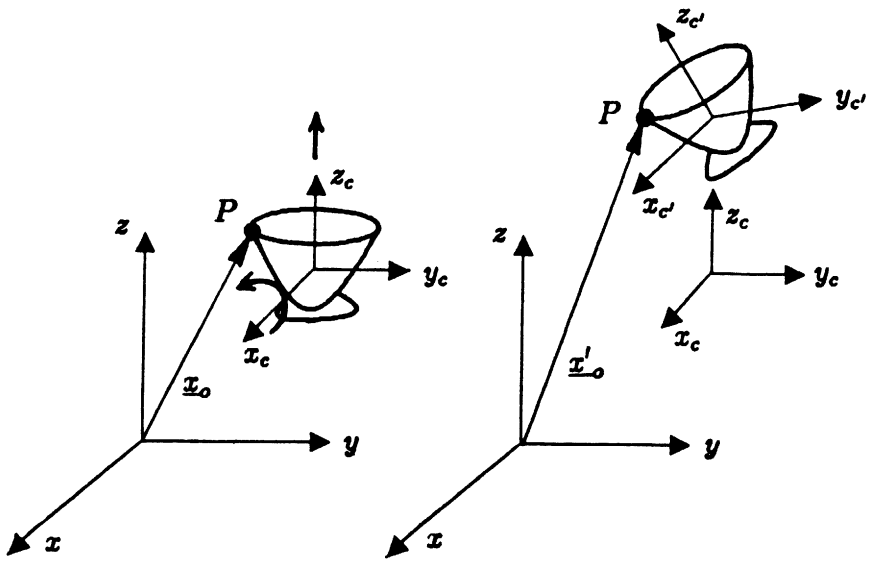



Figure 3.3: Example in using frame transforms

Next, we establish a coordinate frame c′ *fixed* to the cup which initially coincides with frame c. After rotating thru $\theta$ about $\hat{x}_c$ and translating by the distance $h$ along $\hat{z}_c$, the relation of frame c′ to frame c is described by the rotation $[\underline{n}, \theta]$ and the translation $\underline{p}$ where

$$\underline{n} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$$

$$\underline{p} = \begin{bmatrix} 0 & 0 & h \end{bmatrix}^T \ .$$

Let $A_{c'}$ be the transform relating c′ to c. From Equation 3.12 we obtain

$$A_{c'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & h \end{bmatrix} \ .$$

Since the point $P$ is fixed to the cup, its location in frame c′ is the same as its original location in frame c. Denoting the new location by $\underline{x}'$, we have

$$\underline{x}'_{c'} = \underline{x}_c = A_c^{-1} \underline{x}_o \ .$$

Hence, the new location of point $P$ in frame o is given by

$$\underline{x}'_o = A_c A_{c'} \underline{x}'_{c'} = A_c A_{c'} A_c^{-1} \underline{x}_o \ .$$

**Now suppose the cup is securely grasped by an articulated hand with $n$ contact points, performing this transformation for the $n$ points will yield the required contact positions which will place the cup at the desired position and orientation. This procedure forms the basis for generating Cartesian trajectory of objects with an articulated hand.**

# Chapter 4

# Generating Grasp Trajectory

## 4.1 Introduction

Using manipulator contacts to impose kinematic constraints on an object requires that these constraints be self-consistent and consistent with the constraints in the environment. Assuming that the object and the environment geometries are precislely known, consistent constraints can be specified. Obtaining the desired object motion then corresponds to imposing the appropriate kinematic constraints on the object via the contacts.

Articulated robot *hands* are essentially arrangements of fingers; each finger can be viewed as a miniature manipulator having one or more degrees of freedom. The motion of each finger tip is equivalent to the end point motion of an independent manipulator. When an object is grasped by the finger tips, we can view each end point as attached to the object.[1] The motions of the end points then act as kinematic constraints on the possible motions of the object. Manipulation of the object then reduces to the task of imposing an appropriate set of end point constraints such that the possible motions of the object will be uniquely the desired motion.

This chapter studies how to obtain desired Cartesian motion of grasped ob-

[1] Assuming no slip and that rolling at the contacts is negligible.

jects with an articulated hand. It will be assumed that the object and the environment are precisely known, and hence position control of the contacts can be used. The materials will be presented in a general context. The Stanford/JPL Hand [Salisbury 1982] will be used to illustrate materials which are unique to a particular hand. First, we will study the transformations between Cartesian and joint space hand coordinates. Next, a useful coordinate frame called the *grasp frame* will be defined. Finally, the coordinate frame transforms and Cartesian to joint space transforms are combined to translate the desired Cartesian motion of an object into desired motion of the finger joints. By using the grasp frame, we will be able to specify motion of the object in a *body-fixed* coordinate frame.

## 4.2 Hand Kinematic Transforms

The desired motion of an object is almost always specified in terms of Cartesian coordinates. Hence, to use the manipulator contacts as kinematic constraints, the motion of contacts must also be specified in Cartesian space. Given the joint position of the manipulator, to locate the contacts in Cartesian space requires a kinematic transform from joint space to Cartesian space. Conversely, given the Cartesian positions of the contacts, to locate the corresponding joint position requires a *inverse kinematic* transform from Cartesian space to joint space.

### 4.2.1 Dealing with Multiple Solutions

For a *non-redundant* manipulator with $n$ degrees of freedom, we need to derive the transforms which maps the $n$ joint space coordinates into $n$ Cartesian coordinates, and vice versa. The mapping from joint space to Cartesian space will have a unique solution, but the inverse mapping will usually have a finite set of possible solutions. Additional conditions must be imposed on the solutions to obtain a unique mapping. For example, a typical manipulator with six DOF may have eight or more possible configurations which will place the gripper at a desired position and orientation. The path to some configurations may cause

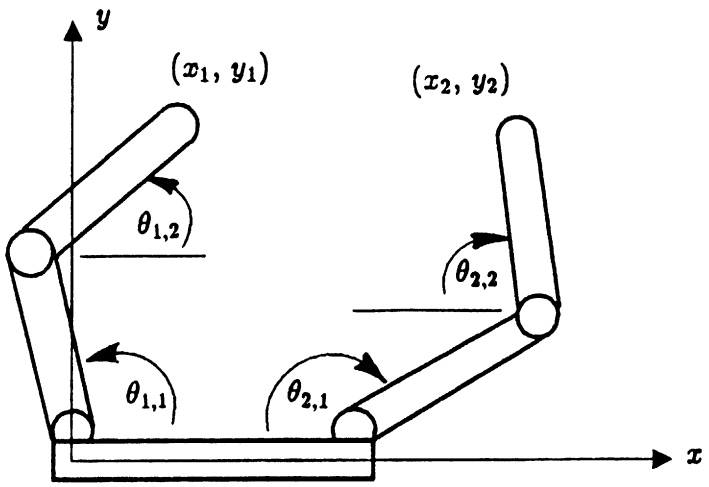



Figure 4.1: Non-redundant articulated hand for planar motion

collisions, while others require awkward movements. An unique configuration must be chosen ahead of time, or sufficient conditions must be imposed such that a unique configuration can be automatically selected.

For a redundant manipulator, the number of DOF in joint coordinates is greater than the number of DOF in Cartesian coordinates. The mapping from joint space to Cartesian space has a unique solution, but the inverse mapping has zero or infinite solutions. Additional constraint equations must be included in the inverse mapping to obtain a finite set of solutions, and then conditions imposed on this set to obtain a unique solution. The additional constraint equations may be chosen to optimize the performance, e.g. even distribution of joint velocities minimize joint torques [Hollerbach and Suh 1985], required power input [Salisbury and Abramowitz 1985], or time of travel [Brooks 1982].

The same principles hold for articulated hands. Consider the planar four DOF non-redundant hand shown in Figure 4.1. Given the four joint angles, there is a unique set of four Cartesian coordinates for the hand, two for each finger end point. Hence, the mapping from joint space to Cartesian space is

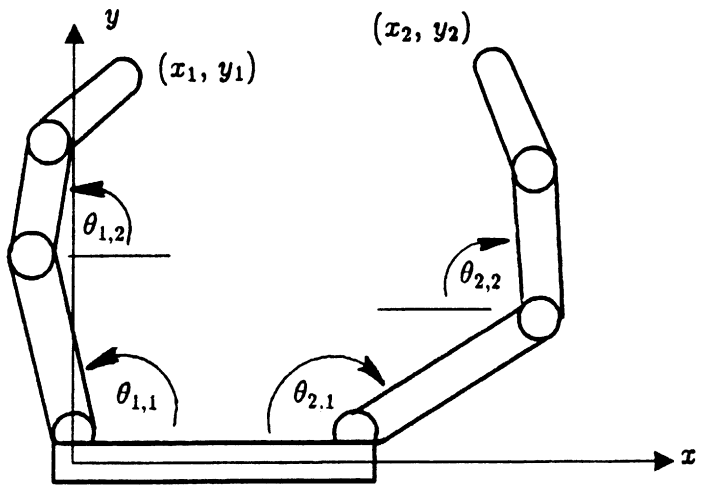



Figure 4.2: Redundant articulated hand for planar motion

given by the single-valued transform

$$\underline{x} = \Lambda(\underline{\theta})$$

where

$$\underline{x} = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 \end{bmatrix}^T$$

$$\underline{\theta} = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{2,1} & \theta_{2,2} \end{bmatrix}^T .$$

However, for a given Cartesian position there are usually four possible combinations of joint positions. The configuration of each finger must be specified to either bend toward the palm or away from the palm in order to obtain a unique joint solution. Hence, the inverse transform

$$\underline{\theta} = \Lambda^{-1}(\underline{x})$$

can have up to four solutions.

For the planar six DOF redundant hand shown in Figure 4.2, the mapping from joint space to Cartesian space is still single-valued. However, there are now inifinite solutions for the inverse transform. This is because there are six

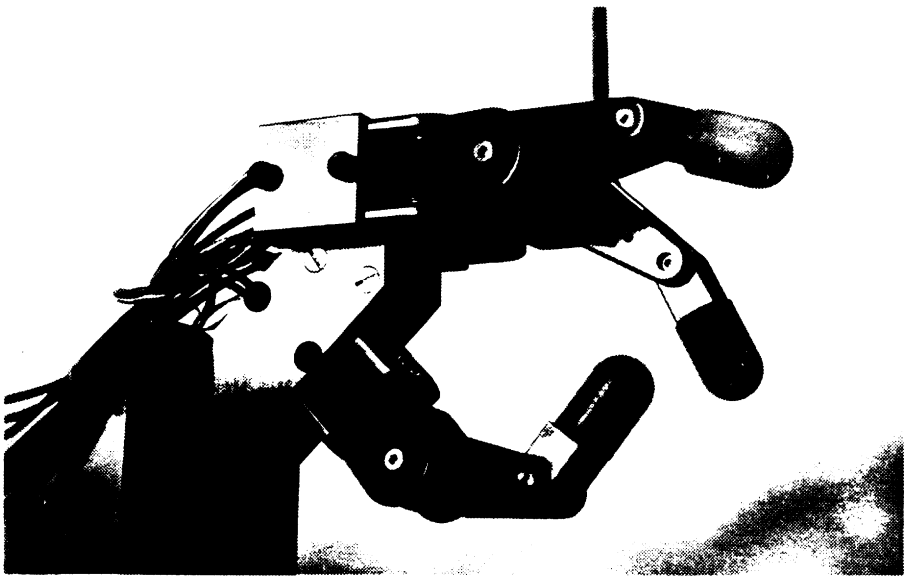


Figure 4.3: The Stanford/JPL Hand

unknowns (the joint angles) but only four constraint equations (the $x$ and $y$ positions of the finger tips). To reduce the possible solutions to a finite set, two additional constraint *equations* are required. For example, we may specify that the ratio of $\theta_{1,1}$ to $\theta_{1,2}$ must be some constant. After the solutions are reduced to a finite set, additional *conditions* are imposed to obtain a unique solution. It is important to distinguish between *constraint equations* and *conditions*. Constraint equations are imposed on the DOF of the manipulator to reduce the possible solutions to a finite set. Conditions are imposed to select a unique solution from the finite set.

### 4.2.2 Transforms for the Stanford/JPL Hand

As an example, we will derive the kinematic transforms for the (non-planar) Stanford/JPL Hand. The Standford/JPL Hand is a nine DOF hand composed of three fingers, each having three DOF. (see Figure 4.3).

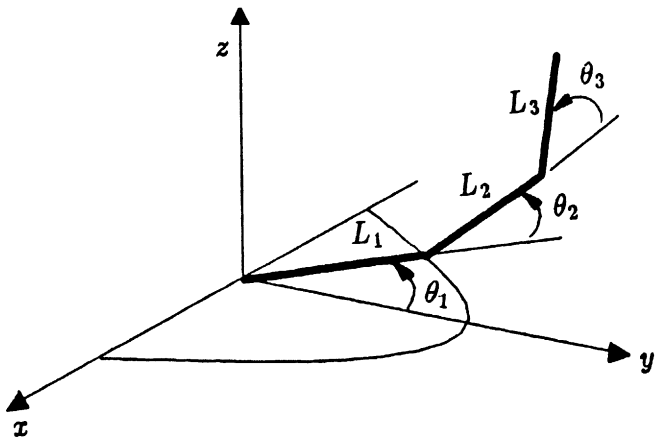



Figure 4.4: Definition of finger coordinates

### Finger Transforms

We will first consider the transformations for a finger in a Cartesian coordinate frame fixed at its base. The Cartesian coordinate of the finger corresponds to the location of the finger tip. The definitions of the joint angles and Cartesian coordinates are shown in Figure 4.4.

The transform from joint space to Cartesian space can be easily found as

$$\Lambda_f(\underline{\theta}) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -(L_1 + L_2\cos\theta_2 + L_3\cos(\theta_2+\theta_3))\sin\theta_1 \\ (L_1 + L_2\cos\theta_2 + L_3\cos(\theta_2+\theta_3))\cos\theta_1 \\ L_2\sin\theta_2 + L_3\sin(\theta_2+\theta_3) \end{bmatrix}. \tag{4.13}$$

The inverse transform involves more complex geometry. Figure 4.5 defines the variables which will be used in the derivation. There are two possible configurations which will place the finger tip at $(x, y, z)$, one corresponds to the finger curling upward and the other corresponds to finger curling downward. From the figure, the first joint angle is simply

$$\theta_1 = \text{atan}\left(\frac{-x}{y}\right). \tag{4.14}$$

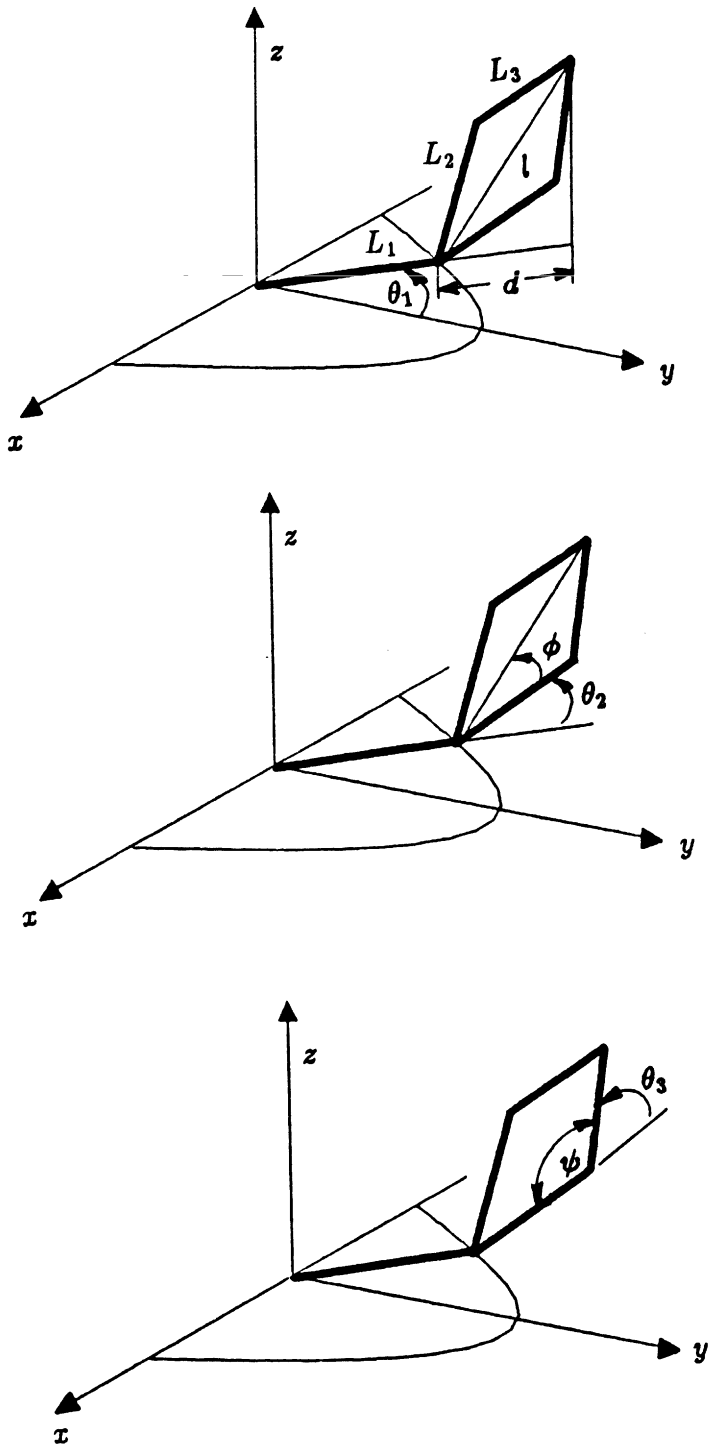



Figure 4.5: Coordinate definitions for derivation of inverse transform

The mechanical design of the hand limits this joint to $|\theta_1| < 90^o$. Otherwise, some positions may have up to four solutions, corresponding to the fingers reaching backwards.

The lengths $d$ and $l$ are found from

$$d = \frac{y}{\cos\theta_1} - L_1$$

$$l = \sqrt{d^2 + z^2} \ .$$

Note that if $l^2 > L_2^2 + L_3^2$, then the desired position is out of reach. The angle $\phi$ is found from the law of cosines

$$\cos\phi = \frac{l^2 + L_2^2 - L_3^2}{2lL_2}$$

and $\theta_2$ is

$$\theta_2 = \text{atan}\left(\frac{z}{d}\right) \mp \phi = \text{atan}\left(\frac{z}{d}\right) \mp \text{acos}\left(\frac{l^2 + L_2^2 - L_3^2}{2lL_2}\right). \qquad (4.15)$$

where $-$ corresponds to the upward curl configuration, and $+$ corresponds to the downward curl configuration. Similarly, we find

$$\cos\psi = \frac{L_2^2 + L_3^2 - l^2}{2L_2L_3}$$

and the solutions for $\theta_3$

$$\theta_3 = \pm(\pi - \psi) = \pm\left(\pi - \text{acos}\left(\frac{L_2^2 + L_3^2 - l^2}{2L_2L_3}\right)\right). \qquad (4.16)$$

where $+$ corresponds to upward curl, and $-$ corresponds to downward curl. Equations 4.14, 4.15, and 4.16, constitutes the inverse transform

$$\underline{\theta} = \Lambda_f^{-1}(\underline{x})$$

which can have from zero to two solutions.[2] When there are two solutions, the condition of whether the finger is curled upward or downward will determine a unique solution.

[2] When there are no limits on the joint 2 and 3 angles, no solution corresponds to the position out of reach; one solution corresponds to links 2 and 3 fully extended.

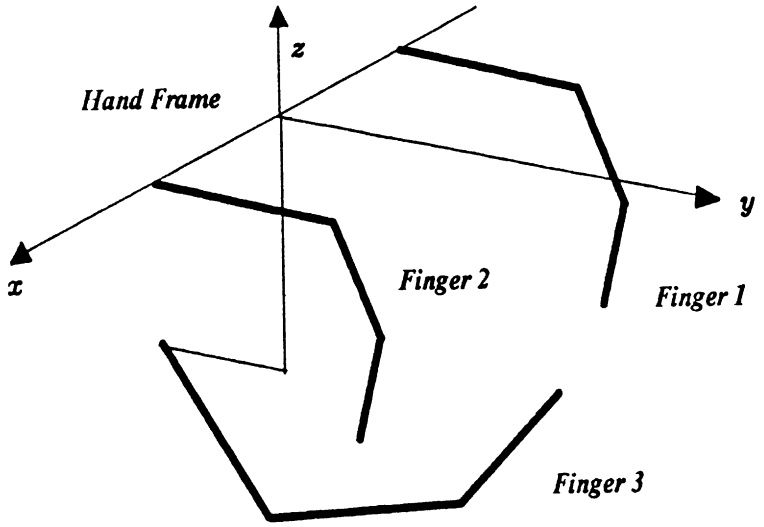



Figure 4.6: Definition of hand coordinate system

### Hand Transforms

The previous transforms were derived with respect to a coordinate system fixed at the base of a finger. To coordinate the motion of the fingers in the hand, the transforms of all the fingers should be derived with respect to a uniform *hand coordinate* system. For the Stanford/JPL Hand, we have chosen the hand coordinate frame as shown in Figure 4.6. The finger transforms in the hand coordinate frame are as previously derived but for a simple shift in origin. The transforms for finger 3 requires an additional rotation of coordinate axes.

The transforms of the individual fingers are combined into a *hand transform* which maps the nine joint coordinates to nine Cartesian coordinates in the hand frame

$$\underline{x} = \Lambda_h(\underline{\theta})$$

where

$$\underline{x} = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \underline{x}_3 \end{bmatrix} \quad \text{and} \quad \underline{\theta} = \begin{bmatrix} \underline{\theta}_1 \\ \underline{\theta}_2 \\ \underline{\theta}_3 \end{bmatrix}.$$

The vectors $\underline{x}_i$ and $\underline{\theta}_i$ are the Cartesian and joint positions of finger $i$, respec-

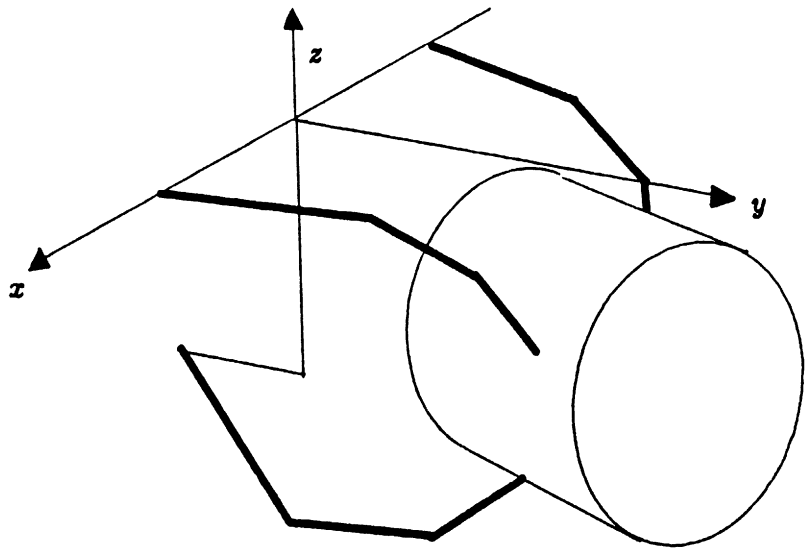



Figure 4.7: Inward curl avoids interference with object

tively. The inverse transform

$$\underline{\theta} = \Lambda_h^{-1}(\underline{x})$$

can have up to eight solutions, since each finger can have two solutions. Curl configurations must be specified for each of the fingers to obtain an unique solution. Instead of specifying the configurations as "curl up" or "curl down", we can now conveniently specify them as curling inwards or outwards from the palm. Since the outward curl configuration may cause the links to interfere with a grasped object, the inward curl should be chosen by default (see Figure 4.7). This will free the programmer from the tedious task of specifying the curl configuration for each of the fingers, unless an outward curl is required.

## 4.3 The Grasp Frame

For a manipulator with a simple gripper, the position and orientation of the grasped object is described by the position and orientation of the hand frame. This is possible because the hand frame is rigidly attached to the object. The desired motion of the object is translated into the desired motion of the hand

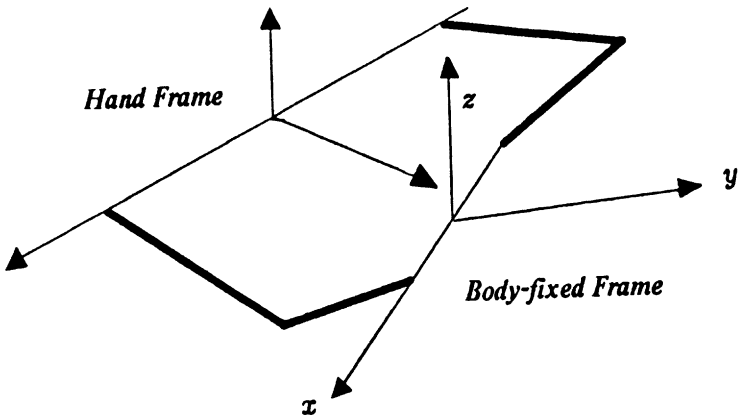



Figure 4.8: Body-fixed frame defined by planar finger contacts

frame. However, when an object is grasped by an articulated hand, the object can be moved and oriented *within* the hand, and hence the position and orientation of the hand frame is insufficient for describing the object. To place an object at the desired position and orientation, we need to manipulate a *body-fixed* coordinate frame, i.e. one rigidly attached to the object.

### 4.3.1 Defining a Body-fixed Frame

Assuming that no slip has occured and that rolling is negligible, finger contacts can be viewed as rigidly attached to the grasped object during any object motion. Therefore, the position and orientation of the contacts can be conveniently used to define a body-fixed coordinate frame. Once this frame is defined, the joint coordinates can be used to compute the location of this frame at any time. An alternative would be to use a vision system to track the object.

As a simple example, consider the planar two-fingered hand shown in Figure 4.8. The two finger tip contacts can be used to define a body-fixed frame. The $z$ direction of the frame is defined as a vector normal to the plane of the hand. The vector parallel to the line connecting the two finger tips is used to

define the $x$ direction. The $y$ direction is defined by the cross product of the $z$ and $x$ vectors. The frame origin is set at the center of the line connecting the two finger tips. The position and orientation of this frame can be easily computed from the joint angles.

To distinguish it from the hand frame, the frame defined by the grasp contacts will be refered to as the *grasp frame*. The grasp frame should always be defined relative to the hand frame. This lends modularity to the hand. When the hand is mounted on a manipulator arm, the absolute position and orientation of the object can be found simply by multiplying the grasp frame matrix by the hand frame matrix computed from the arm position. Hence, by defining the grasp frame relative to the hand frame, we can write hand software which are independent of the arm used. With this in mind, we shall henceforth use the hand frame as the "absolute" coordinate frame for defining positions and orientations. The configurations and motions of objects in the hand frame can always be transformed to a global manipulator frame by a simple matrix multiplication.

### 4.3.2 Frame Definition for Stanford/JPL Hand

The definition of the grasp frame should be chosen based on usefulness and ease of computation. A useful choice of frame origin is the centroid of the area or volume enclosed by the contacts. However, for ease of computation, one may simply choose a point contact as the frame origin. Readily defined vectors should also be exploited to avoid complexity.

To illustrate the computation of a grasp frame, we consider the contacts of the Stanford/JPL Hand. The contacts with an object are idealized as point contacts at the finger tips. Figure 4.9 shows a grasp frame definition using these contact points. The three points define a triangle in space. The vector from finger 1 to finger 2 defines the $x$ direction of the frame. The outward pointing vector normal to the plane of the triangle defines the $y$ direction. The $z$ direction is defined by the cross product of the $x$ and $y$ vectors. The frame origin is set at the centroid of the triangle.

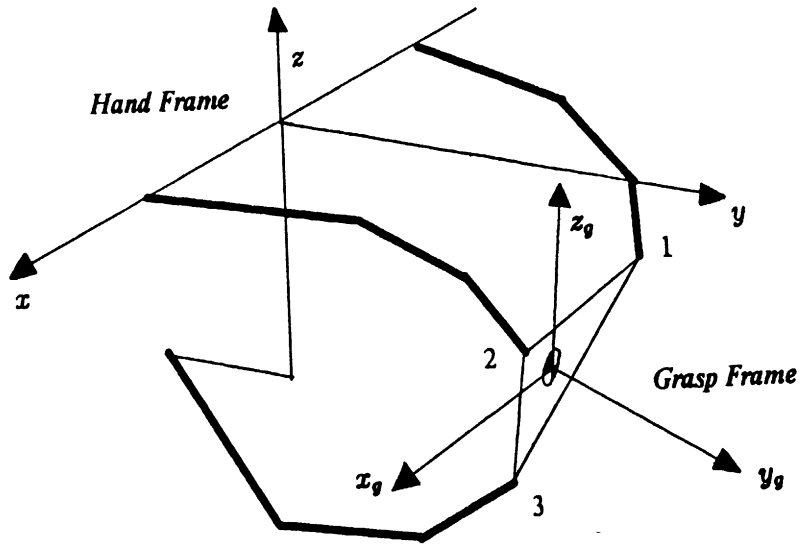



Figure 4.9: Grasp frame definition for Stanford/JPL Hand

Given the joint position $\underline{\theta}$ of the hand, we first find the corresponding Cartesian position of the contact points from the kinematic transform

$$\underline{x} = \Lambda_h(\underline{\theta}) \ .$$

Define $\underline{v}_1$ as the vector pointing from finger 1 to finger 2, and $\underline{v}_2$ as the vector pointing from finger 1 to finger 3

$$\underline{v}_1 = \underline{x}_2 - \underline{x}_1$$

$$\underline{v}_2 = \underline{x}_3 - \underline{x}_1 \ .$$

Let $\hat{\underline{x}}$, $\hat{\underline{y}}$, and $\hat{\underline{z}}$ denote the unit vectors of the grasp frame expressed in the hand frame. We have

$$\hat{\underline{x}} = \frac{\underline{v}_1}{|\underline{v}_1|}$$

$$\hat{\underline{y}} = \frac{\underline{v}_1 \times \underline{v}_2}{|\underline{v}_1 \times \underline{v}_2|}$$

$$\hat{\underline{z}} = \hat{\underline{x}} \times \hat{\underline{y}} \ .$$

If at this time we decide to simply place the origin of the grasp frame at one of the contact points, say $\underline{x}_1$, the grasp frame is then related to the hand frame by

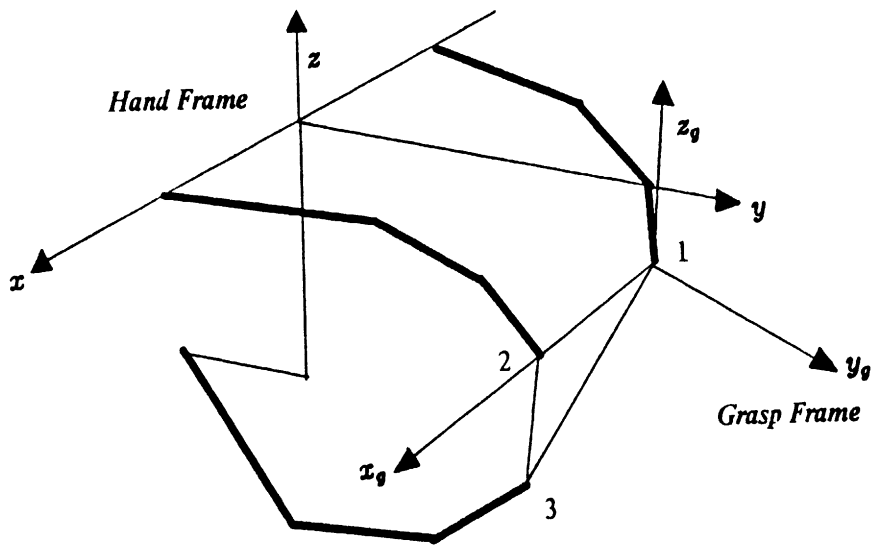



Figure 4.10: Alternate grasp frame definition

the transform matrix (see Equation 3.2)

$$A_{g'} = \begin{bmatrix} \hat{\underline{x}} & \hat{\underline{y}} & \hat{\underline{z}} & \underline{x}_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{4.17}$$

The coordinate system for this grasp frame is shown in Figure 4.10. With respect to this frame, the triangle lies in the $x$-$z$ plane, and one of its sides coincides with the $x$ axis. The position of the centroid in this frame can be easily found. Let the position of the centroid in this frame be

$$\underline{x}'_c = \begin{bmatrix} x'_c & 0 & z'_c \end{bmatrix}^T .$$

Then the position of the centroid in the hand frame is given by

$$\underline{x}_c = A_{g'} \underline{x}'_c .$$

This is now substitued for $\underline{x}_1$ in Equation 4.17 to obtain the grasp frame with origin at the centroid of the triangle

$$A_g = \begin{bmatrix} \hat{\underline{x}} & \hat{\underline{y}} & \hat{\underline{z}} & \underline{x}_c \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{4.18}$$

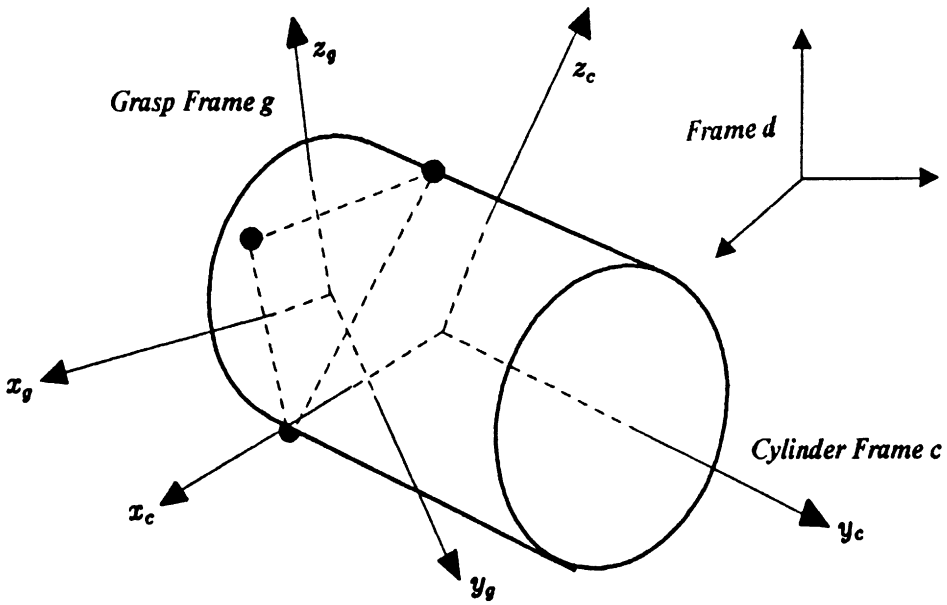



Figure 4.11: Using the grasp frame

Note that in this derivation we have used the standard representation and homogeneous representation of positions interchagebly. This mixture of representations will be used to avoid verbosity. Hence, if a position vector is multiplied by a transform matrix, it is taken to mean that its homogeneous representation is multiplied by the matrix.

### 4.3.3 Using the Grasp Frame

The grasp frame provides a body-fixed coordinate frame computable from the joint positions. However, this frame may not be appropriate for describing the position and orientation of a grasped object. For example, consider the cylinder shown in Figure 4.11. The grasp frame **g** is not as appropriate for describing the position and orientation of the cylinder as frame **c**. If the grasp points on the cylinder are known, then frame **c** can be defined relative to the grasp frame **g**. Then frame **c** can be located at any time by multiplying the relative frame **c** matrix by the grasp frame matrix.

To illustrate the usefulness of the grasp frame, we will consider the problem of moving the can to the desired position and orientation of frame **d**. The

"absolute" coordinate frame is understood to be the hand frame. Given the joint position $\underline{\theta}$, we begin by finding the finger tip positions

$$\underline{x} = \Lambda_h(\underline{\theta}) \ .$$

The grasp frame $A_g(\underline{x})$ is then computed. Let the can frame relative to the grasp frame be described by the matrix $A_{c'}$. Then the position and orientation of the can frame is described by

$$A_c = A_g A_{c'}$$

and the contact locations in the can frame are found from

$$\underline{x}_c = A_c^{-1}\,\underline{x} \ .$$

Note that we have used the simple matrix multiplication to denote the transformation in which every point in $\underline{x}$ is multiplied by $A_c^{-1}$. This notation will be used to avoid verbosity. Now assume that the can frame is moved to coincide with frame d, defined by the matrix $A_d$. Since the location of the contact points relative to the body-fixed can frame do not change, the location of the contacts in the absolute frame is

$$\underline{x'} = A_d\,\underline{x}_c \ .$$

Hence, to move the can from configuration c to d would require moving the fingers from $\underline{x}$ to $\underline{x'}$ given by

$$\underline{x'} = A_d[A_g A_{c'}]^{-1}\,\underline{x} \ .$$

The required joint position is then found from the inverse transform

$$\underline{\theta'} = \Lambda_h^{-1}(\underline{x'}) \ .$$

This example illustrates the simplicity of defining an alternate body-fixed coordinate frame and obtaining the desired position and orientation of this frame. The grasp frame provides the necessary link between the hand coordinates and the position and orientation of a grasped object.

## 4.4 Generating a Trajectory

The use of frame transforms has made it possible to compute the required *final* positions of the fingers corresponding to a desired object position and orientation. If the individual fingers were to move in straight line motion to their respective final positions, the shape of the grasp will alter during the motion, e.g. the distance between the grasp points will change. This will cause either dropping or crushing of the manipulated object. The computation of the final positions assumes that the contacts are *rigidly attached* to the object. Hence, during every instant of object motion, the relative locations of the contacts must remain constant. The solution to this problem requires knowledge of the position and orientation of the object at every instant of the motion. The positions of the contacts are then transformed using the body-fixed coordinate frame into positions in the absolute frame.

To obtain the position and orientation of the object at every instant of the motion requires a *specification of motion.* That is, instead of specifying *where* the object is to be moved, we must specify *how.* This is philosophically very different from motion specifications for ordinary manipulators. When using a manipulator, the desired gripper position and orientation is translated directly into goal joint positions. The joints are then simply servoed to their final positions. Specification of *how* the gripper is to be moved to its final position and orientation is not necessary. Paul [1975,1979] and Taylor [1979] have both investigated the problem of obtaining Cartesian trajectories of the gripper, but the fundamental motion command for a manipulator is still a desired position and orientation. For an articulated hand, however, the fundamental motion command is the desired *trajectory* of an object. A goal position and orientation must be specified in terms of a trajectory. In this section we will study how object motions are specified and how to translate the desired motion into trajectory of contact points.

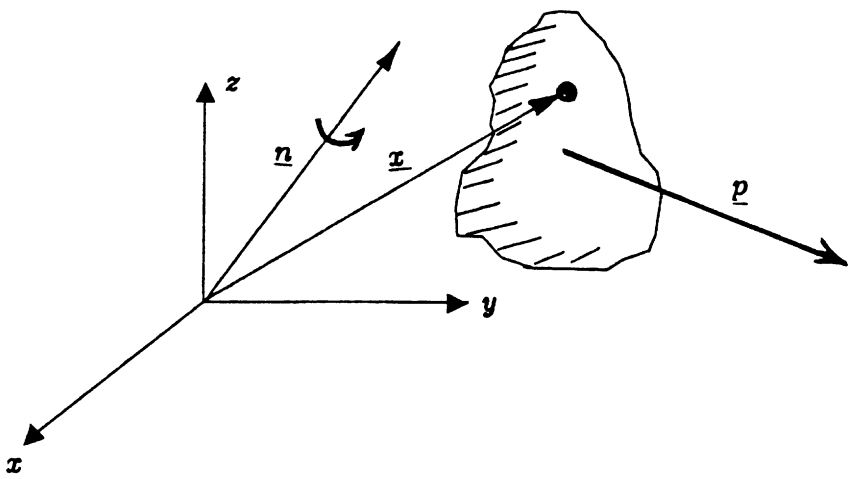



Figure 4.12: Describing motion of an object

### 4.4.1 Describing Motion of Objects

Consider the motion of a rigid object as shown in Figure 4.12. We wish to rotate the object about an axis $\underline{n}$ by the angle $\theta_d$. Let $T$ be the time duration of the rotation and let $\underline{x}(0)$ locate a point on the object before the rotation. Then, for uniform motion, the point at time $t$ is located by

$$\underline{x}(t) = \begin{bmatrix} R(\underline{n}, \theta(t)) & \underline{0} \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} \underline{x}(0)$$

where $R(\underline{n}, \theta)$ is given by Equation 3.12 and

$$\theta(t) = \frac{t}{T}\theta_d \ .$$

Similarly, if we wish to translate the object along the vector $\underline{p}_d$ in time T, the point is located by

$$\underline{x}(t) = \begin{bmatrix} I & \underline{p}(t) \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} \underline{x}(0)$$

where $I$ is the identity matrix and

$$\underline{p}(t) = \frac{t}{T}\underline{p}_d \ .$$

Placing the object in an arbitrary position and orientation can be accomplished by a rotation followed by a translation. However, dividing the motion into two sequences is both time consuming and unnatural. The same position and orientation can be obtained by a single motion. Let the required rotation be $[\underline{n}, \theta_d]$ and the translation be $\underline{p}_d$, we define the corresponding "straight line" motion by the point trajectory

$$\underline{x}(t) = \begin{bmatrix} R(\underline{n}, \theta(t)) & \underline{p}(t) \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} \underline{x}(0) \ . \tag{4.19}$$

This motion corresponds to superimposing the translation on the rotation. We shall call the set $[\underline{n}, \theta_d, \underline{p}_d]$ a *move specification*, or simply a *move spec*. Equation 4.19 defines the motion corresponding to the move spec with duration $T$.

### 4.4.2 Specifying Motion

The above formulation provides the basis for determining required contact trajectories corresponding to a desired object motion. We will now study how trajectories corresponding to some basic tasks are obtained.

#### Motion in Alternate Frames

Often we wish to specify motion with respect to a frame which is not the absolute coordinate frame. For example, the rotation shown in Figure 4.13 is obtained by specifying rotation about the $z$ axes of frame o. To generate motions with respect to an arbitrary frame, we can simply obtain the trajectory of the contacts in that frame and then transform it back to the absolute frame.

Let the move $[\underline{n}, \theta_d, \underline{p}_d]$ be specified with respect to a frame $A_o$, and let $\underline{x}(0)$ be the absolute position of the contacts at time $t = 0$. The contact positions in frame $A_o$ are then given by

$$\underline{x}_o(0) = A_o^{-1} \underline{x}(0) \ .$$

When the object is rotated and translated with respect to the axes in frame $A_o$,

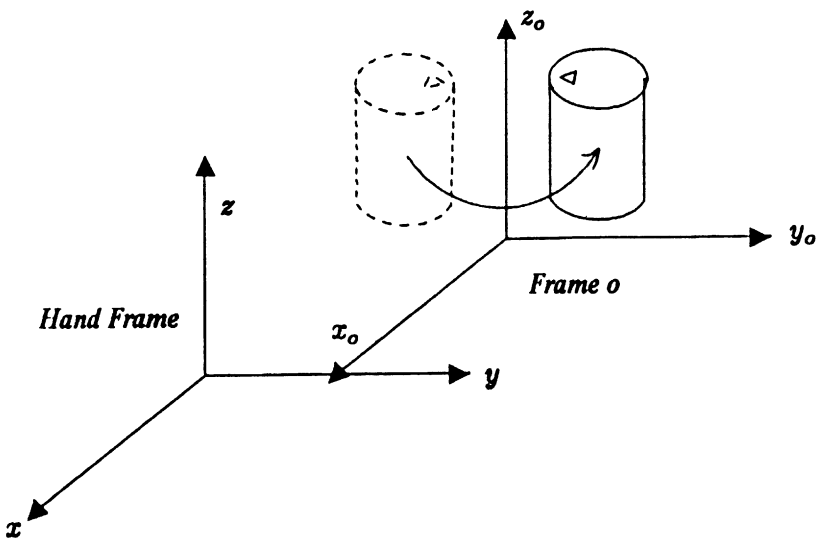



Figure 4.13: Specifying motion with respect to another frame

the trajectory of the contacts in frame $A_o$ is

$$\underline{x}_o(t) = \begin{bmatrix} R(\underline{n}, \theta(t)) & \underline{p}(t) \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} \underline{x}_o(0) \ .$$

The required trajectory of the contacts in the absolute frame is then easily found from

$$\underline{x}(t) = A_o \, \underline{x}_o(t) \ .$$

Hence, we have

$$\underline{x}(t) = A_o \begin{bmatrix} R(\underline{n}, \theta(t)) & \underline{p}(t) \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} A_o^{-1} \, \underline{x}(0). \tag{4.20}$$

The contact trajectory $\underline{x}(t)$ can then be transformed into the required joint trajectory $\underline{\theta}(t)$ by the inverse transform

$$\underline{\theta}(t) = \Lambda_h^{-1} \left( \underline{x}(t) \right) \ .$$

### Object Centered Motions

A body-fixed coordinate frame defines an origin and a set of axes which describes the position and orientation of an object. Often we wish to rotate or

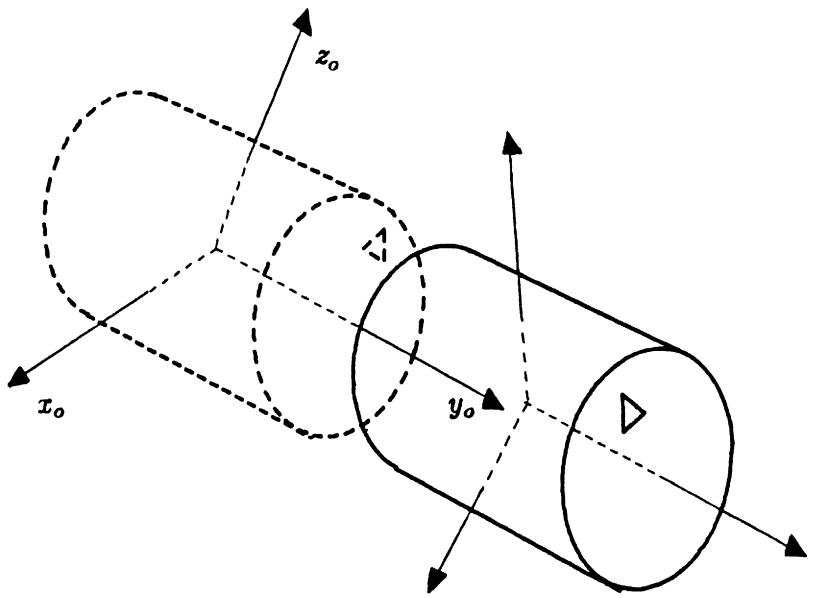



Figure 4.14: Screw motion is generated with respect to body-fixed frame

translate an object along one of its own axes. For example, the screw motion shown in Figure 4.14 is obtained by specifying simultaneous rotation about and translation along the $y$ axis. This class of *object centered motion* can be easily obtained by the use of the grasp frame.

Let the body-fixed frame of an object be described by $A_{o'}$ relative to the grasp frame, and let the move $[\underline{n}, \theta_d, \underline{p}_d]$ be specified with respect to the body-fixed frame. We begin by evaluating the current grasp frame $A_g(\underline{x}(0))$. The body-fixed frame is then

$$A_o = A_g(\underline{x}(0))A_{o'} \ .$$

The trajectory of the contacts are then computed in this frame and transformed back to the absolute frame, as given by Equation 4.20.

### Positioning and Orienting Objects

As discussed previously, a trajectory specification is the fundamental motion command for an articulated hand, while for an ordinary manipulator it is a desired position and orientation of the object/gripper. The position and orientation commands for a manipulator can be used to generate desired object

trajectories [Taylor 1979]. Similarly, the trajectory commands for an articulated hand can be used to obtain desired positions and orientations of objects. We will now consider how to transform a position and orientation specification into a trajectory specification.

Recall that the relation between two frames can be described by an equivalent rotation about an unit vector $\underline{n}$ followed by a translation $\underline{p}$ of the frame origin. That is, one frame can be brought to coincide with another by a single rotation followed by a translation. This offers a well defined motion path for moving a body-fixed frame to a desired position and orientation. However, as previously discussed, dividing the motion into two sequences is both time consuming and unnatural. Hence, we will adopt the "straight line" motion defined by Equation 4.19. The straight line motion between the two frames can be envisioned as superimposing the rotation on the translation.[3]

The equivalent rotation $[\underline{n}, \theta]$ is used to describe the *relation* between two frames. Hence, if we wish to move a body-fixed coordinate frame from $A_o$ to some frame $A_p$, then we must first express $A_p$ as a frame relative to $A_o$. Let $\underline{x}$ be a point in the absolute frame. Let its location in frame $A_o$ be given by $\underline{x}_o$, and its location in frame $A_p$ be given by $\underline{x}_p$. Then we have

$$\underline{x} = A_o \underline{x}_o = A_p \underline{x}_p$$

or

$$\underline{x}_o = [A_o^{-1} A_p] \underline{x}_p \ .$$

Since the matrix $[A_o^{-1} A_p]$ transforms $\underline{x}_p$ to $\underline{x}_o$, $A_p$ must be related to $A_o$ by the matrix

$$A_{p'} = A_o^{-1} A_p \ .$$

The first three columns of this matrix is used in Equations 3.8 thru 3.11 to find the equivalent rotation $[\underline{n}, \theta_d]$, and the last column is the origin translation vector $\underline{p}_d$. The task of moving the body-fixed frame from $A_o$ to $A_p$ now reduces to generating the trajectory $[\underline{n}, \theta_d, \underline{p}_d]$ in frame $A_o$.

[3]Paul [1975] defines "straight line" motion as translation of the frame origin coupled with two composite rotations. This definition would require computation of rotations about two axes.

## 4.5 Interpolation in Joint Space

In the previous section we have studied how to obtain the required trajectory of the contacts corresponding to a given task. The contact trajectory is given by a continous time function $\underline{x}(t)$. The corresponding continous time joint trajectory is found from the inverse transform

$$\underline{\theta}(t) = \Lambda^{-1}\,\underline{x}(t) \ .$$

Computer control of maniplator involve specifying a desired joint position or force for each interval of servo cycles. Hence, we can only specify goal positions for isolated points in time. To compute a transformation for every servo cycle usually cannot be done in real time. The computations must be performed prior to the execution of the trajectory, i.e. the joint trajectory must be *preplanned.* For manipulations in which trajectory decisions are made based on sensory input, it is important to minimize the time the manipulator waits for the planning to be completed. Therefore, only a selected number of intermediate trajectory points should be computed. The number of computations should be the minimum required to satisfy some "fineness" criteria for the motion.

To illustrate the computations, we will consider a trajectory specified with respect to the absolute frame. Trajectories in other frames can be obtained by a constant matrix multiplication. Let the trajectory $[\underline{n}, \theta_d, \underline{p}_d]$ in time interval $T$ be divided into $N$ uniform segments, then the position of the contacts at each knot point is given by

$$\underline{x}(t_k) = \begin{bmatrix} R(\underline{n}, \theta(t_k)) & \underline{p}(t_k) \\ 0\ \ 0\ \ 0 & 1 \end{bmatrix} \underline{x}(0)$$

where

$$t_k = k\,\frac{T}{N} \ .$$

Define the incremental rotation angle and translation vector

$$\Delta\theta = \frac{1}{N}\theta_d$$

$$\Delta \underline{p} = \frac{1}{N} \underline{p}_d \ .$$

Since rotations are composed by multiplying rotation matrices, we can write

$$R(t_k) = R^k(\underline{n}, \Delta\theta)$$

and for the translation

$$\underline{p}(t_k) = k\, \Delta \underline{p} \ .$$

These two equations offer a convenient algorithm for computing the positions of trajectory knot points. The corresponding knot point in joint space is

$$\underline{\theta}(t_k) = \Lambda_h^{-1}\, \underline{x}(t_k) \ .$$

This corresponds to $N$ joint trajectory segments, each with duration $T/N$. The joints can be commanded to simply move from $\underline{\theta}(t_k)$ to $\underline{\theta}(t_{k+1})$ in a straight line in joint space, or quadratic curve fitting may be used to smooth out the transitions between the segments.

The interpolations in joint space will result in intermittent deviations of the contact trajectory from the desired path. Philosophically, to maintain the grasp shape would require the continuous time contact trajectory $\underline{x}(t)$ be faithfully executed. Interpolations in joint space would result in distortions of the grasp shape and hence lead to either dropping or crushing of the manipulated object. In pratice, we can command a slightly smaller grasp shape than the actual shape of the object. The mechanical compliance of the fingers are used to absorb the contact force variations due to distortions in the commanded grasp shape.

With respect to manipulator Cartesian motion, Taylor [1979] has suggested a method for determining the number of knot points required. He noted that the maximum deviation from a Cartesian path usually occur at or near the midpoint of a joint trajectory segment. Therefore, the Cartesian position and orientation corresponding to the midpoint of a segment can be used as a convenient measurement of the accuracy of the motion. The position and orientation computed from the joint segment midpoint are compared with the desired Cartesian path midpoint. If the deviations exceeds some specified bounds, then the segment is

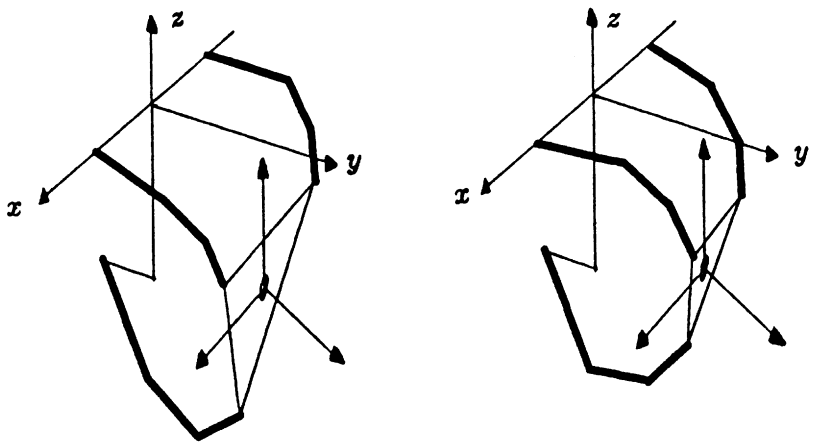



Figure 4.15: Two grasp shapes have same grasp frame

divided in two and the computations repeated for the midpoints of these two segments.

Similar algorithms can be used to determine the required number of knot points for an articulated hand. The position and orientation of an object at the joint segment midpoint can be computed by evaluating the corresponding grasp frame. However, the grasp frame may not be an accurate indicator of the correctness of the motion. For example, the two grasp shapes shown in Figure 4.15 have the same grasp frame. Excessive contact forces will result from mistaking one configuration as satisfactory. Hence, we may also wish to evaluate the distances between the contacts and check if the variations exceeded some specified bounds.

*This empty page was substituted for a blank page in the original document.*

# Chapter 5

# Grasp Stiffness Control

## 5.1 Introduction

Philosophically, position control of an articulated hand is feasible only when the shape of the object and the environment are perfectly known. The motions of the contacts can then be used as a set of consistent kinematic constraints on the object. Specifying the positions of $n$ contact points is equivalent to $3n$ constraint equations on the object.[1] Since an object only has six DOF, the solution to the position and orientation of the object is *over constrained* when there are more than two contact points.[2] Using coordinate transforms, we were able to specify contact motions for which all constraint equations are consistent. However, interpolations in joint space *inevitably* result in intermittent inconsistencies in the constraints. In practice, we can use the mechanical compliance of the fingers to absorb the interaction forces due to attempts to impose inconsistent constraints. That is, the passive compliance of the fingers is exploited to ensure that the actual motions of the finger tips are geometrically compatible with that of the object. Allowing the passive compliance of the fingers to ensure geometric compatibility corresponds to absorbing the kinematic inconsistencies

[1] Assuming three-freedom constraints such as point contacts with friction.

[2] Actually, the requirement that the distance between two contacts must remain constant also imposes a constraint equation. Hence, we can only specify two coordinates of the second contact point without over constraining the object.

through deformation of the fingers. When the fingers do not have a sufficient range of elastic deformations, permanent damage may occur. If the object is also compliant, then deformation of the object will occur as well.

Since position control will inevitably result in kinematically inconsistent contact motions, force control is a necessity, at least philosophically, for articulated hands. Force control corresponds to imposing force constraints at the contacts instead of kinematic constraints. The motion of the contacts is specifically allowed to adapt to the object motion. Force control also eliminates possible kinematic inconsistencies due to constraints in the enviroment. As the contacts comply to the object motion, the object motion will comply to the enviromental constraints.

There are several force control strategies suitable for a manipulator with a single endpoint, as described in Chatper 2. For these manipulators the object is securely grasped by a simple gripper. The force at the endpoint is controlled to produce a desired net force on the object such that the motion of the object satisfies the constraints in the environment. For an articulated hand, multiple endpoints must be coordinated to produce the desired *internal force* which maintains a stable grasp, as well as produce the desired net force on the object.

The *stiffness control* strategy presented by Salisbury [1980] is particularly suitable for articulated hands. For manipulator endpoint motion, restoring forces are exerted proportional to the deviation of the position from a desired nominal trajectory. When applied to an articulated hand, in addition to exerting net forces proportional to the object trajectory deviation, internal grasp forces can be exerted in proportion to the deviation of the distances between the grasp points [Salisbury 1982]. Hence, the object behaves as if attached to a set of interconnected springs at each contact. Instead of enforcing the intermittently inconsistent contact positions, the inconsistent positions are treated as *nominal* positions. The deviations of the actual positions from the joint-interpolated values are now absorbed by the *active* spring instead of the mechanical compliance of the finger.

The stiffness strategy is sometimes also referred to as the *generalized spring* strategy because it imparts to a manipulator endpoint the characteristics of a multi-dimensional spring. This is one in a class of *causal* strategies in which the manipulator acts as an impedance. An analogous *generalized damper* strategy imparts to the manipulator the charateristics of a dashpot. Using this strategy, forces and moments are exerted in proportion to the deviations from a nominal translational and rotational velocity. The *impedance control* strategy [Hogan 1984] combines both of the generalized spring and damper characteristics with a generalized inertia; additional forces are exerted in proportion to the acceleration. Stiffness control can be used only when there is sufficient mechanical damping in the manipulator or in the environment to ensure a stable closed-loop system. In general, active damping should be included to guarantee stability of the combined manipulator-environment system.

This chapter will begin with an analysis of the stiffness matrix which defines the force/displacement relationship. Then we will consider the implementation of stiffness control using an articulated hand. When the expected deviations are small, it will be possible to *pre-compute* a joint space stiffness corresponding to a desired Cartesian stiffness. The results are then generalized to include full impedance control.

## 5.2 The Stiffness Matrix

Stiffness control involves implementing a desired relationship between force and displacement. Restoring forces and moments are exerted on an object as a function of the displacement from some desired nominal position and orientation. We will consider a linear Cartesian force/displacement relation defined by

$$\underline{\mathcal{F}} = -K\,\Delta\underline{\mathcal{X}} \tag{5.21}$$

where $\underline{\mathcal{F}}$ is the generalized force vector which includes both translational and rotational forces, and $\Delta\underline{\mathcal{X}}$ is the generalized displacement vector which includes

both position and orientation, i.e.

$$\underline{\mathcal{F}} = \begin{bmatrix} f_x & f_y & f_z & m_x & m_y & m_z \end{bmatrix}^T$$

and

$$\Delta\underline{\mathcal{X}} = \underline{\mathcal{X}} - \underline{\mathcal{X}}_{nom} = \begin{bmatrix} \Delta x & \Delta y & \Delta z & \Delta\theta_x & \Delta\theta_y & \Delta\theta_z \end{bmatrix}^T .$$

$K$ is a general 6×6 *stiffness matrix* which defines the desired force/displacement relation. The coordinate frame in which the net force and moments on the object are defined is referred to as the *compliance frame.* The point at which a pure force can be exerted without causing rotation is called the *compliance center.* We envision an object at its nominal configuration is rigidly attached to a frame which initially coincides with the compliance frame. As the object is displaced relative to the compliance frame, the position and orientation of the atached frame is described by the vector $\Delta\underline{\mathcal{X}}$. The origin of the displaced frame is located by the first three elements of the vector. The orientation of the frame corresponds to the rotation $[\underline{n}, \Delta\theta]$ where

$$\Delta\theta \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \Delta\theta_x \\ \Delta\theta_y \\ \Delta\theta_z \end{bmatrix} .$$

Alternately, we can view the vector $\Delta\underline{\mathcal{X}}$ as the effective move specification in the compliance frame which corresponds to the object displacement.

### 5.2.1 Diagonal Matrices

Active stiffness control of the manipulator imparts to the manipulated object a controlled compliance. That is, the displacement of the object from its nominal position will be proportional to the force exerted on it by the environment. The use of passive compliance in assembly tasks was analyzed by Drake [1977]. His work led to the development of the Remote Center Compliance (RCC) which is used for chamfered assembly tasks. The RCC is basically a four-dimensional spring which allows lateral and angular realignment of the constrained object. For a peg-in-hole insertion task, the ideal location of the compliance center was

shown to be at the mouth of the hole [Whitney 1982]. However, since the RCC is a passive device, the compliance center must remain fixed with respect to the peg. Hence, the compliance center is placed near the tip of the peg where the desired relation is satisfied approximately. By using active stiffness control, the effective compliance center can be easily shifted to remain near the hole as the peg is inserted. This will be shown in a programming example in Chapter 7.

Using active stiffness control, a full six-dimensional RCC can be implemented by the diagonal stiffness matrix

$$K = \begin{bmatrix} k_x & 0 & 0 & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 & 0 & 0 \\ 0 & 0 & k_z & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{\theta_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{\theta_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{\theta_z} \end{bmatrix}$$

defined at the compliance center. The current four-dimensional RCC has no translational or rotational compliance about the vertical axis, which corresponds to

$$k_z = k_{\theta_z} = \infty$$

Diagonal stiffness matrices appear frequently in literature because its effects are easy to visualize. The use of a diagonal stiffness matrix is also sufficient for many assembly tasks. However, the capabilities of active stiffness control far exceeds implementing the simple decoupled force/displacement relation. In what follows, the physical significance of non-diagonal stiffness matrices will be investigated.

### 5.2.2 Non-diagonal Matrices

Consider an arbitrary stiffness matrix $K$ with distinct eigenvalues. A matrix $P$ may be found which diagonalizes the stiffness matrix by the following transformation

$$K_D = P^{-1} K P \tag{5.22}$$

or

$$K = PK_DP^{-1} \tag{5.23}$$

where $K_D$ is a diagonal matrix with the eigenvalues of $K$ on the diagonal, and $P$ is a matrix with columns composed of the corresponding eigenvectors. Substitution into Equation 5.21 yields

$$\underline{\mathcal{F}} = -PK_DP^{-1}\Delta\underline{\mathcal{X}} \ .$$

Let the vectors $\underline{\mathcal{F}}_D$ and $\underline{\mathcal{X}}_D$ be defined by

$$\underline{\mathcal{F}} = P\,\underline{\mathcal{F}}_D$$

$$\underline{\mathcal{X}} = P\,\underline{\mathcal{X}}_D$$

then we can write

$$\underline{\mathcal{F}}_D = -K_D\,\Delta\underline{\mathcal{X}}_D \ .$$

This is a decoupled force/displacement relation. The decoupled force and displacement directions are along the column vectors of $P$, i.e. the eigenvectors. The eigenvalues and eignevectors may be real or complex. The eigenvector corresponding to a real eigenvalue is always real, and that corresponding to a complex eigenvalue is always complex.

Before we begin the analysis of eigenvalues and eigenvectors, the following definitions are useful

**Definition 1** *A matrix $W$ is said to be a unitary matrix if*

$$\overline{W}^TW = W\overline{W}^T = I$$

*where $\overline{W}$ denotes a matrix whose elements are complex conjugates of the elements of matrix $W$, and $I$ denotes the identity matrix.*

**Definition 2** *A matrix $H$ is said to be a normal matrix if*

$$\overline{H}^TH = H\overline{H}^T \ .$$

It is a fact from linear algebra that only normal matrices can be diagonalized by a unitary transformation of the form

$$H_D = \overline{W}^T H W$$

where $H_D$ is a diagonal matrix with the eigenvalues of $H$ on the diagonal. Furthermore, if $H$ is real, then $H_D$ and $W$ will be real.

Since the elments of a stiffness matrix $K$ are always real, we have

$$\overline{K}^T = K^T$$

and hence the matrix will be normal if

$$K^T K = K K^T \ .$$

From this definition, we see that only symmetric stiffness matrices are normal, i.e. $K = K^T$. Therefore, *only symmetric stiffness matrices can be diagonalized by a unitary transformation.*

### Symmetric Matrices

We now consider a symmetric stiffness matrix $K$ and investigate the significance of a unitary transformation. Since $K$ is real, the diagonal eigenvalue matrix $K_D$ and the corresponding unitary transformation matrix $W$ will be real. Hence, the definition of unitary matrix yields

$$W^T W = W W^T = I$$

or

$$W^T = W^{-1} \ .$$

Therefore, the unitary transformation can be written as

$$K_D = W^{-1} K W \ .$$

Comparison with Equation 5.22 shows that the unitary transformation matrix corresponds to the matrix $P$. The property

$$P^T P = P P^T = I \tag{5.24}$$

is obtained by normalizing the eigenvectors in the matrix. Let $\underline{v}_i$ and $\underline{v}_j$ be two column vectors in the matrix $P$, then Equation 5.24 states that

$$\underline{v}_i^T \underline{v}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

i.e. the eigenvectors are orthogonal. Some physical insights can be gained by considering a simple 3×3 translational stiffness matrix. For the symmetric 3×3 stiffness matrix, we have

$$\underline{\mathcal{F}} = \begin{bmatrix} f_x & f_y & f_z \end{bmatrix}$$

$$\Delta\underline{\mathcal{X}} = \begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix}$$

$$P = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \underline{v}_3 \end{bmatrix} .$$

The set of orthogonal eigenvectors transforms the coordinates into the decoupled force/displacement relations

$$\underline{\mathcal{F}}_D = -K_D \, \Delta\underline{\mathcal{X}}_D$$

where $\underline{\mathcal{F}}_D$ and $\underline{\mathcal{X}}_D$ are defined by

$$\underline{\mathcal{F}} = P\underline{\mathcal{F}}_D$$

$$\underline{\mathcal{X}} = P\underline{\mathcal{X}}_D \ .$$

Since $P^T P = I$, the inverse of $P$ is simply its transpose. Recalling that the inverse of a rotation matrix is also its transpose, we see that $P$ is essentially a rotation matrix describing the orientation of the decoupled axes with respect to the original axes. Hence, the symmetric stiffness can be viewed as obtained from a simple rotation of a diagonal stiffness. The decoupled stiffnesses, or eigenvalues of $K$, are completely analogous to the principal stresses in solid mechanics and the principal moments of inertia in rigid body dynamics. The eigenvectors then correspond to the principal axes. We will refer to these axes as *principal translational stiffness axes*. If the 3×3 $K$ matrix is a symmetric rotational stiffness matrix, then corresponding *principal rotational stiffness axes* are obtained.

If we construct a 6×6 stiffness matrix with decoupled translational and rotational stiffnesses, the orientation of the principal translational axes and principal rotational axes can be specified independently. This is accomplished by a $K$ matrix such that

$$K = \begin{bmatrix} K_x & 0 \\ 0 & K_\theta \end{bmatrix}$$

where $K_x$ and $K_\theta$ are 3×3 translational and rotational stiffness matrices, respectively, and each 0 represents a 3×3 matrix of zeros. The $K_x$ matrix is used to orient the translatonal stiffness axes, and $K_\theta$ is used to orient the rotational stiffness axes. The required $K_x$ and $K_\theta$ are found by reversing the unitary transformation (see Equation 5.23). Hence,

$$K_x = R_x K_{x,d} R_x^T$$

$$K_\theta = R_\theta K_{\theta,d} R_\theta^T$$

where $K_{x,d}$ and $K_{\theta,d}$ are the desired diagonal principal stiffnesses; $R_x$ and $R_\theta$ are the rotation matrices describing the desired orientation of the principal translational and rotational stiffness axes, respectively.

For a general symmetric 6×6 stiffness matrix $K$, the principal directions are in genral not purely translational or rotational. The three translational displacements and the three rotational displacements can be viewed as a generalized displacement in six-dimensional space. Although the translation axes coincide with the rotation axes in three-dimensional space, they are orthogonal axes in six-dimensional space. The normalized eigenvector matrix $P$ then represent a six-dimensional transformation matrix which describes the orientation of the six-dimensional principal axes. After diagonalizing the stiffness matrix, six decoupled real force/displacement relations are obtained

$$f_i = -k_i \, \Delta x_i \quad i = 1, 2, \cdots, 6$$

where $k_i$ is the $i^{th}$ eigenvalue of $K$. In order for *restoring* force to be exerted for a given displacement $\Delta x_i$, the stiffness $k_i$ must be positive. If $k_i$ is negative, then a displacement will cause a force which will further compound the displacement.

Any general displacement having components along the $i^{th}$ eigenvector will be unstable. If $k_i$ is zero, then no force is exerted along the $i^{th}$ eigenvector, i.e. that direction is perfectly compliant. Therefore, to ensure asymptotic stability, all eigenvalues of $K$ must be positive, i.e. the stiffness matrix must be positive definite.

The requirement of positive-definitness can also be obtained from energy considerations. An elastic system is stable if work must be done to cause a displacement from its nominal configuration. Hence, if the nominal configuration is taken to have zero energy, then the energy stored in the displaced system must be positive. Given a generalized spring with stiffness $K$, the energy stored corresponding to the displacement $\Delta\underline{\chi}$ is

$$E = \frac{1}{2}\Delta\underline{\chi}^T K\,\Delta\underline{\chi} \ .$$

The energy $E$ will be positive for all displacement $\Delta\underline{\chi} \neq \underline{0}$ if and only if the matrix $K$ is positive definite.[3]

A positive definite stiffness matrix is ensured when the principal stiffnesses are selected to be positive. The required stiffness matrix $K$ corresponding to a desired diagonal stiffness $K_D$ can be easily found by reversing the unitary transformation, i.e. from

$$K = PK_DP^T$$

where $P$ is a six-dimensional rotation matrix describing the desired orientation of the principal axes.

Whether stiffness control of a manipulator implements a stiffness or a compliance depends on how we view the interaction with the environment. With respect to external kinematic constraints, the spring acts as a stiffness, i.e. force is exerted in proportion to displacement. With respect to external forces, the spring acts as a compliance, i.e. displacement is proportional to the external force. For the symmetric stiffness matrix, the *compliance* relations are given by

$$\Delta x_i = \frac{1}{k_i} f_i \qquad i = 1, 2, \cdots, 6$$

[3]This is the definition of positive definitness. The requirement that all eigenvalues be positive is a sufficient test for positive definitness.

where $f_i$ is now the *external* force in the direction of the $i^{th}$ eigenvector.

### Non-symmetric Matrices

Non-symmetric stiffness marices do not satisfy the definition for a normal matrix. Therefore, it cannot be diagonalized by a unitary transform, i.e. the eigenvectors of non-symmetric matrices are not orthogonal. This means the stiffness matrix cannot be decoupled by a simple rotation of axes, whether in three-dimensional or six-dimensional space. Furthermore, the eigenvalues and eigenvectors are no longer guaranteed to be real. The diagonal matrix $K_D$ and the eigenvector matrix $P$ will in general contain both real and complex elements. As eigenvalues come in complex conjugate pairs, the corresponding eigenvectors are also complex conjugates. That is, if $k_1$ and $k_2$ are a pair of complex eigenvalues, then they are of the form

$$k_1 = k_\nu + jk_\mu$$

$$k_2 = k_\nu - jk_\mu$$

and the corresponding eigenvectors are of the form

$$\underline{v}_1 = \underline{\alpha} + j\underline{\beta}$$

$$\underline{v}_2 = \underline{\alpha} - j\underline{\beta}$$

By definition the eigenvectors satisfies the following relation

$$K\,\underline{v}_1 = \underline{v}_1 k_1 = (\underline{\alpha} + j\underline{\beta})(k_\nu + jk_\mu) = (k_\nu\underline{\alpha} - k_\mu\underline{\beta}) + j(k_\mu\underline{\alpha} + k_\nu\underline{\beta})$$

$$K\,\underline{v}_2 = \underline{v}_2 k_2 = (\underline{\alpha} - j\underline{\beta})(k_\nu - jk_\mu) = (k_\nu\underline{\alpha} - k_\mu\underline{\beta}) - j(k_\mu\underline{\alpha} + k_\nu\underline{\beta})$$

Therefore, we can write

$$\frac{1}{2}K(\underline{v}_1 + \underline{v}_2) = K\,\underline{\alpha} = k_\nu\underline{\alpha} - k_\mu\underline{\beta}$$

$$\frac{1}{2j}K(\underline{v}_1 - \underline{v}_2) = K\,\underline{\beta} = k_\mu\underline{\alpha} + k_\nu\underline{\beta}$$

Hence, the decoupled complex equation

$$K\begin{bmatrix} \underline{v}_1 & \underline{v}_2 \end{bmatrix} = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 \end{bmatrix}\begin{bmatrix} k_\nu + jk_\mu & 0 \\ 0 & k_\nu - jk_\mu \end{bmatrix}$$

has been transformed into the coupled *real* equation

$$K\begin{bmatrix} \underline{\alpha} & \underline{\beta} \end{bmatrix} = \begin{bmatrix} \underline{\alpha} & \underline{\beta} \end{bmatrix} \begin{bmatrix} k_\nu & k_\mu \\ -k_\mu & k_\nu \end{bmatrix}$$

The eigenvector matrix $P$ can be transformed into a purely real matrix $P^*$ by transforming each pair of complex eigenvectors $[\underline{\alpha} + j\underline{\beta}, \underline{\alpha} - j\underline{\beta}]$ into a corresponding pair of real vectors $[\underline{\alpha}, \underline{\beta}]$ as shown above. The stiffness matrix $K$ will then be transformed by the $P^*$ matrix into a purely real matrix. For example,

$$K_D^* = [P^*]^{-1} K P^* = \begin{bmatrix} k_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{\nu,3} & k_{\mu,3} & 0 & 0 \\ 0 & 0 & -k_{\nu,3} & k_{\nu,3} & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{\nu,4} & k_{\mu,4} \\ 0 & 0 & 0 & 0 & -k_{\mu,4} & k_{\nu,4} \end{bmatrix} . \tag{5.25}$$

Just as the response of linear dynamic systems can be decoupled into first and second order responses, we can decouple the generalized spring into "first and second order springs". The "first order spring" will be referred to as a *simple spring*, and the "second order spring" will be referred to as a *complex spring*. The decoupled real force/displacement relations will then be either of the form

$$f_i = -k_i \, \Delta x_i \tag{5.26}$$

or of the form

$$\begin{bmatrix} f_i \\ f_j \end{bmatrix} = - \begin{bmatrix} k_\nu & k_\mu \\ -k_\mu & k_\nu \end{bmatrix} \begin{bmatrix} \Delta x_i \\ \Delta x_j \end{bmatrix} \tag{5.27}$$

where the decoupled coordinate axes are along the column vectors of $P^*$.

The characteristics of the simple spring described by Equation 5.26 is well understood. We will now examine the characteristics of the complex spring described by Equation 5.27. Consider the complex spring

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = - \begin{bmatrix} k_\nu & k_\mu \\ -k_\mu & k_\nu \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} \tag{5.28}$$

The energy stored in the deflected spring is

$$E = \frac{1}{2}\,\Delta\underline{x}^T K\,\Delta\underline{x} = \frac{1}{2}(k_\nu\,\Delta x_1^2 + k_\nu\,\Delta x_2^2) \ .$$

Therefore, from the standpoint of energy storage, the complex spring acts as two simple springs with stiffness $k_\nu$, the real part of the complex stiffness. Equation 5.28 shows that deflection in one direction can cause force to be exerted in another direction. More insight can be gained by considering the *compliance* of the spring. Inverting the stiffness matrix, we obtain the compliance relation

$$\begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \frac{1}{k_\nu^2 + k_\mu^2} \begin{bmatrix} k_\nu & -k_\mu \\ k_\mu & k_\nu \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

where $f_1$ and $f_2$ now represent external forces, and $\Delta x_1$ and $\Delta x_2$ are the resulting deflections.

We see that force in one direction can cause deflections in another direction. The term $k_\mu$, the imaginary part of the complex stiffness, provides the cross-coupling. This behavior can be useful in moving objects across obstacles, e.g. climbing over a wall. In essence, the term $k_\nu$ is used for its value as a stiffness, and the term $k_\mu$ is used for its value as a cross-coupling compliance.

This analysis shows the potential uses of non-symmetric stiffness matrices. The required stiffness matrix $K$ can be easily obtained from the desired simple and complex stiffnesses. The matrix $K$ is found by reversing the transformation of Equation 5.25

$$K = P^* K_D^* [P^*]^{-1}$$

where $P^*$ is selected from the desired directions of the simple and complex stiffness axes in six-dimensional space.

## 5.3 Force Application

The basis of stiffness control is pure force control, i.e. the exertion of the desired forces on an object. Stiffness control is essentially specifying the desired forces as functions of the measured displacements. In this section we will consider

the issues of exerting the desired forces on an object. Much of the materials here were presented by Salisbury [1982] in his works on hand kinematics and force analysis. We will develop these ideas from a more heuristic approach. Coordinate transformations are then added to allow the specification of forces in arbitrary coordinate frames.

### 5.3.1 Generalized Contact Force

The kinematic transform for a manipulator yields a functional relation of the form

$$\underline{\chi} = \Lambda(\underline{\theta})$$

where $\underline{\chi}$ are the Cartesian coordinates of the manipulator endpoint corresponding to the joint coordinates $\underline{\theta}$. The Jacobian $J$ is defined as the matrix which transforms the joint velocities to the Cartesian velocities

$$\dot{\underline{\chi}} = J\,\dot{\underline{\theta}}$$

where the elements of $J$ are given by

$$J_{i,j} = \frac{\partial \chi_i}{\partial \theta_j} \ .$$

The endpoint Cartesian force $\underline{\mathcal{F}}$ is related to the joint coordinate force $\underline{\tau}$ by the transpose of the Jacobian

$$\underline{\tau} = J^T\,\underline{\mathcal{F}} \ . \tag{5.29}$$

Each finger of an articulated hand can be viewed as an independent manipulator. The velocities of the contacts are related to the finger joint velocities by the individual finger Jacobians. Hence, for finger $i$, we have

$$\dot{\underline{x}}_i = J_i\,\dot{\underline{\theta}}_i \ .$$

Similarly, the forces at the contacts are related to the finger joint torques by

$$\underline{\tau}_i = J_i^T\,\underline{f}_i \ .$$

The *generalized contact force* $\underline{F}$ is defined as a vector composed of the individual finger contact force vectors. For a hand with $m$ fingers, we have

$$\underline{F} = \begin{bmatrix} \underline{f}_1 & \underline{f}_2 & \cdots & \underline{f}_m \end{bmatrix}^T$$

To exert a desired contact force $\underline{F}$, the required joint torque vector $\underline{\tau}$ is computed from the hand Jacobian $J$

$$\underline{\tau} = J^T \underline{F} \tag{5.30}$$

where

$$\underline{\tau} = \begin{bmatrix} \underline{\tau}_1 & \underline{\tau}_2 & \cdots & \underline{\tau}_m \end{bmatrix}^T$$

and

$$J^T = \begin{bmatrix} J_1^T & 0 & 0 & 0 & \cdot & \cdot \\ 0 & J_2^T & 0 & 0 & \cdot & \cdot \\ 0 & 0 & \cdot & & & \\ 0 & 0 & & \cdot & & \\ \cdot & & & & \cdot & \\ \cdot & & & & & J_m^T \end{bmatrix} .$$

### 5.3.2 Generalized Grasp Force

We define the *generalized grasp force* $\underline{\mathcal{F}}$ as a vector with the first six elements being the net translational and rotational forces exerted on the object and the remaining elements being the magnitude of the *internal grasp forces.* For a hand with $n$ elements in the contact force vector, the generalized grasp force is

$$\underline{\mathcal{F}} = \begin{bmatrix} f_x & f_y & f_z & m_x & m_y & m_z & g_1 & \cdots & g_{n-6} \end{bmatrix}^T .$$

As the Jacobian matrix $J$ relates the joint torques to the generalized contact force, we can construct a $n \times n$ *grasp matrix* $G$ which relates the contact force to the generalized grasp force

$$\underline{F} = G^T \underline{\mathcal{F}} . \tag{5.31}$$

To exert a desired grasp force on an object, we first compute the required contact force from Equation 5.31. Then the required joint torques are computed from

the hand Jacobian $J$ as in Equation 5.30. Therefore, the required joint toque corresponding to a desired grasp force $\underline{\mathcal{F}}$ is

$$\underline{\tau} = J^T G^T \underline{\mathcal{F}} = [GJ]^T \underline{\mathcal{F}} \ . \tag{5.32}$$

Comparing with Equation 5.29 for an ordinary manipualtor, we see that the matrix $[GJ]$ of an articulated hand is analogous to the Jacobian matrix of the manipulator. Similarly, the velocity relation for the hand is

$$\dot{\underline{\mathcal{X}}} = GJ\,\dot{\underline{\theta}} \tag{5.33}$$

where $\dot{\underline{\mathcal{X}}}$ is the generalized velocity corresponding to the generalized force $\underline{\mathcal{F}}$. Hence, the first six elements of $\dot{\underline{\mathcal{X}}}$ are the translational and rotational velocities of the grasped object.

Unlike the Jacobian, $G$ or $G^T$ cannot be found directly from geometry. This is because forces are exerted by parallel links rather than serial links. Hence, we can only geometrically compute the transform which maps $\underline{F}$ to $\underline{\mathcal{F}}$, but not in the reverse direction as in Equation 5.31. From Equation 5.31, we have

$$\underline{\mathcal{F}} = G^{-T}\,\underline{F}$$

where $G^{-T}$ denotes the inverse of the transpose of $G$. This equation allows the solution for the $G^{-T}$ transform to be found by projection; $G^T$ or $G$ are then found by matrix inversion. Therefore, to have a defined grasp matrix requires that $G^{-T}$ be non-singular. This means that the $n-6$ internal grasp forces must be chosen such that the last $n-6$ rows of the $G^{-T}$ matrix are mutually independent and independent of the first six rows. From the viewpoint of specifying constraints, specifying $\underline{\mathcal{F}}$ corresponds to imposing $n$ constraint equations on the variable $\underline{F}$. Each row of $G^{-T}$ corresponds to a linear constraint equation, which must be independent in order for a unique solution to exist.

### 5.3.3 Grasp Matrix for the Stanford/JPL Hand

As an example, we will compute the grasp matrix for the Stanford/JPL Hand. The definitions for the variables used in the derivation is shown in Figure 5.1.

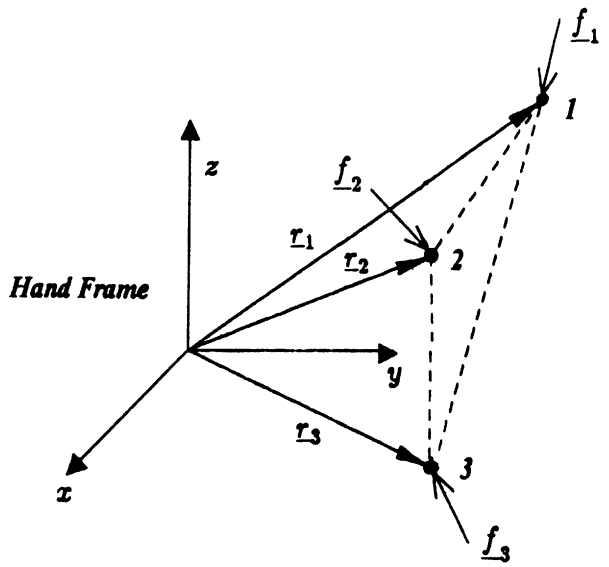



Figure 5.1: Coordinate definitions for deriving grasp matrix

Idealizing the contacts as point contacts with friction, each finger is capable of exerting three translational forces. Hence

$$\underline{f_i} = \begin{bmatrix} f_{ix} & f_{iy} & f_{iz} \end{bmatrix}^T$$

and the generalized contact force is

$$\underline{F} = \begin{bmatrix} \underline{f_1} & \underline{f_2} & \underline{f_3} \end{bmatrix}^T \ .$$

Since the generalized contact force has nine elements, the generalized grasp force will also be a nine element vector. The first three elements are the net translational force $\underline{f}$ exerted on the object

$$\underline{f} = \underline{f_1} + \underline{f_2} + \underline{f_3} \ . \tag{5.34}$$

The next three elements in the grasp force vector are the net moment $\underline{m}$ exerted on the object. Let the position of contact $i$ be $\underline{r_i}$, we have

$$\underline{m} = \underline{r_1} \times \underline{f_1} + \underline{r_2} \times \underline{f_2} + \underline{r_3} \times \underline{f_3} \tag{5.35}$$

where

$$\underline{r_i} = \begin{bmatrix} r_{ix} & r_{iy} & r_{iz} \end{bmatrix}^T \ .$$

Since there are a total of nine force degrees of freedom for the hand, we must specify three more force constraint equations. Otherwise there will be inifinite contact force solutions which will result in the desired net force on the object. This *force redundancy* allows us to control useful internal grasp forces. We define the three internal grasp forces to be the "squeeze force" along the edges of the grasp triangle. Defining the *unit* vectors pointing from contact $i$ to contact $j$ as $\underline{r}_{ij}$, the internal grasp force vector is

$$\underline{g} = \begin{bmatrix} \underline{g}_{12} \\ \underline{g}_{13} \\ \underline{g}_{23} \end{bmatrix} = \begin{bmatrix} \underline{r}_{12} \cdot \underline{f}_1 - \underline{r}_{12} \cdot \underline{f}_2 \\ \underline{r}_{13} \cdot \underline{f}_1 - \underline{r}_{13} \cdot \underline{f}_3 \\ \underline{r}_{23} \cdot \underline{f}_2 - \underline{r}_{23} \cdot \underline{f}_3 \end{bmatrix} . \tag{5.36}$$

A positive value for $g_{ij}$ indicates squeezing the object between contacts $i$ and $j$. Recall the transform relation

$$\underline{\mathcal{F}} = G^{-T} \underline{F}$$

or

$$\begin{bmatrix} \underline{f} \\ \underline{m} \\ \underline{g} \end{bmatrix} = G^{-T} \begin{bmatrix} \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_3 \end{bmatrix} .$$

Comparison with Equations 5.34, 5.35, and 5.36 shows that the matrix $G^{-T}$ is

$$G^{-T} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & -r_{1z} & r_{1y} & 0 & -r_{2z} & r_{2y} & 0 & -r_{3z} & r_{3y} \\ r_{1z} & 0 & -r_{1x} & r_{2z} & 0 & -r_{2x} & r_{3z} & 0 & -r_{3x} \\ -r_{1y} & r_{1x} & 0 & -r_{2y} & r_{2x} & 0 & -r_{3y} & r_{3x} & 0 \\ r_{12x} & r_{12y} & r_{12z} & -r_{12x} & -r_{12y} & -r_{12z} & 0 & 0 & 0 \\ r_{13x} & r_{13y} & r_{13z} & 0 & 0 & 0 & -r_{13x} & -r_{13y} & -r_{13z} \\ 0 & 0 & 0 & r_{23x} & r_{23y} & r_{23z} & -r_{23x} & -r_{23y} & -r_{23z} \end{bmatrix} \tag{5.37}$$

This matrix is then inverted to obtain $G^T$. This is a derivation of the grasp matrix given by Salisbury and Craig [1982].

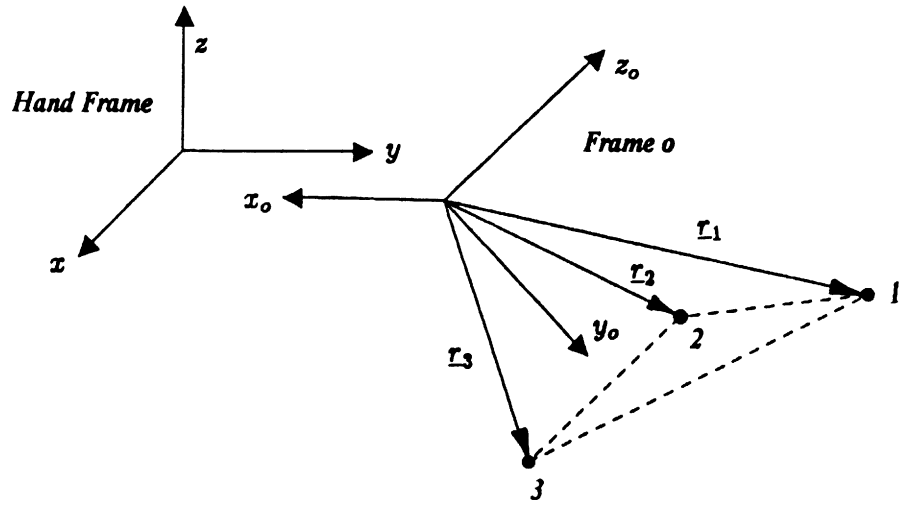



Figure 5.2: Computing the grasp matrix in an alternate frame

### 5.3.4 Specifying Force in Alternate Frames

The previous derivations have assumed that the grasp force $\underline{f}$ was specified in the absolute hand frame. This resulted in the simple relation for the required joint torques

$$\underline{\tau} = J^T G^T \underline{f} \ .$$

When $\underline{f}$ is specified with respect to some frame $A_o$, the grasp matrix must be computed in that frame. That is, the contact position vector $\underline{r}_i$ in Equation 5.37 become the relative position of the contact in frame $A_o$ (see Figure 5.2). These positions can be computed from the absolute positions $\underline{x}_i$ as

$$\underline{r}_i = A_o^{-1} \underline{x}_i \ .$$

The required contact force vector computed from $G^T \underline{f}$ are vectors expressed in frame $A_o$. They are easily transformed to force vectors in the absolute frame by multiplying by the rotation matrix $R_o$ of frame $A_o$, i.e.

$$\underline{f}_i = R_o \underline{f}_{i,o} \ .$$

The contact force vector in the absolute frame can then be transformed into the required torques by the Jacobian matrix. For the Stanford/JPL Hand, this

yields

$$\underline{\tau} = J^T \mathcal{R}_o G^T \underline{\mathcal{F}} \tag{5.38}$$

where

$$J = \begin{bmatrix} J_1 & 0 & 0 \\ 0 & J_2 & 0 \\ 0 & 0 & J_3 \end{bmatrix}$$

and

$$\mathcal{R}_o = \begin{bmatrix} R_o & 0 & 0 \\ 0 & R_o & 0 \\ 0 & 0 & R_o \end{bmatrix} .$$

Therefore, the matrix $[J^T \mathcal{R}_o]$ is the effective Jacobian transpose in frame $A_o$. Substituting the effective Jacobian $[\mathcal{R}_o^T J]$ into the Jacobian of Equation 5.33, we obtain the velocity relation in frame $A_o$

$$\dot{\underline{\mathcal{X}}} = G \mathcal{R}_o^T J \dot{\underline{\theta}}$$

where $\dot{\underline{\mathcal{X}}}$ is now the generalized grasp velocity specified in frame $A_o$.

## 5.4 Stiffness Control

Using the force analysis developed in the previous section, we are ready to implement the stiffness relation

$$\underline{\mathcal{F}} = -K \Delta \underline{\mathcal{X}}$$

where $\underline{\mathcal{F}}$ now includes the internal grasp force, e.g. force between grasp points, and $\Delta \underline{\mathcal{X}}$ now include the corresponding internal displacement, e.g. distance between grasp points. It is also desirable to exert a nominal bias force $\underline{\mathcal{F}}_b$ to maintain positive contact between the fingers and the object or between the object and the environment. Hence, we will implement the more general relation

$$\underline{\mathcal{F}} = -K \Delta \underline{\mathcal{X}} + \underline{\mathcal{F}}_b \ . \tag{5.39}$$

If the displacement $\Delta \underline{\mathcal{X}}$ can be found, then the required torque for the corresponding $\underline{\mathcal{F}}$ can be computed as shown in the previous section, with the grasp matrix evaluated in the compliance frame.

### 5.4.1 The Joint Stiffness Matrix

The first six elements of the vector $\Delta\underline{\chi}$ are the translational and rotational displacements of a body-fixed frame from the nominal position and orientation. The computation of the first six displacements requires evaluating the grasp frame from the finger positions and comparing the actual body-fixed frame with the desired nominal frame. The remaining elements in $\Delta\underline{\chi}$ requires computing the internal positions and comparing with the desired nominal positions. The computational requirements makes this approach infeasible in real time. However, when the displacements are small, we can use approximations which produce a more efficient algorithm. Assuming the compliance frame has the same orientation as the absolute frame, the velocity relation in the compliance frame is given by Equation 5.33 as

$$\dot{\underline{\chi}} = GJ\,\dot{\underline{\theta}} \ .$$

For small perturbations, we can write

$$\Delta\underline{\chi} \approx GJ\,\Delta\underline{\theta}$$

where $\Delta\underline{\theta}$ is the deviation in joint position from the nominal coordinate. Substituting into Equation 5.39, we obtain

$$\underline{\mathcal{F}} = -KGJ\,\Delta\underline{\theta} + \underline{\mathcal{F}}_b \ .$$

The required torque is

$$\underline{\tau} = J^T G^T\,\underline{\mathcal{F}} = -J^T G^T KGJ\,\Delta\underline{\theta} + J^T G^T\,\underline{\mathcal{F}}_b \ .$$

Defining the *grasp Jacobian* as the matrix

$$J_g = GJ \ ,$$

the control law can be written compactly as

$$\underline{\tau} = -J_g^T K J_g\,\Delta\underline{\theta} + J_g^T\,\underline{\mathcal{F}}_b \ . \tag{5.40}$$

The name "grasp Jacobian" is used for the matrix because of its similarity to the role of the Jacobian in manipulator force control. Alternately, we can write the equation as

$$\underline{\tau} = -K_\theta \, \Delta\underline{\theta} + \underline{\tau_b} \tag{5.41}$$

where $K_\theta$ is a *joint stiffness matrix* defined by

$$K_\theta = J_g^T K J_g \tag{5.42}$$

and $\underline{\tau_b}$ is the bias torque defined by

$$\underline{\tau_b} = J_g^T \, \underline{\mathcal{F}_b} \ . \tag{5.43}$$

Equation 5.41 is a joint level stiffness control law which relates the joint torques directly to joint displacements. The joint stiffness matrix is in essence the proportional gain matrix of a multi-input/multi-output joint position control system. This algorithm is computationally efficient, and, more importantly, the joint stiffness matrix can be pre-computed based on the desired nominal position. Pre-computing the joint stiffness matrix will eliminate evaluation of the Jacobian and the grasp matrix in real time, hence allowing higher servo rates and better control.

### 5.4.2 Stiffness Control in Alternate Frames

The joint stiffness matrix and bias torque given by Equations 5.42 and 5.43 are valid if the compliance frame have the same orientation as the absolute frame. If the stiffness relation is specified in an alternate frame, the Jacobian must be modified to account for the orientation of the force and displacement vectors. For the Stanford/JPL Hand, this is accomplished by replacing the Jacobian by the modified Jacobian $[\mathcal{R}_o^T J]$, where $\mathcal{R}_o$ is the generalized transformation matrix which transforms the contact force vectors in the compliance frame into force vectors in the absolute frame. Hence, the grasp Jacobian becomes

$$J_g = G\mathcal{R}_o^T J$$

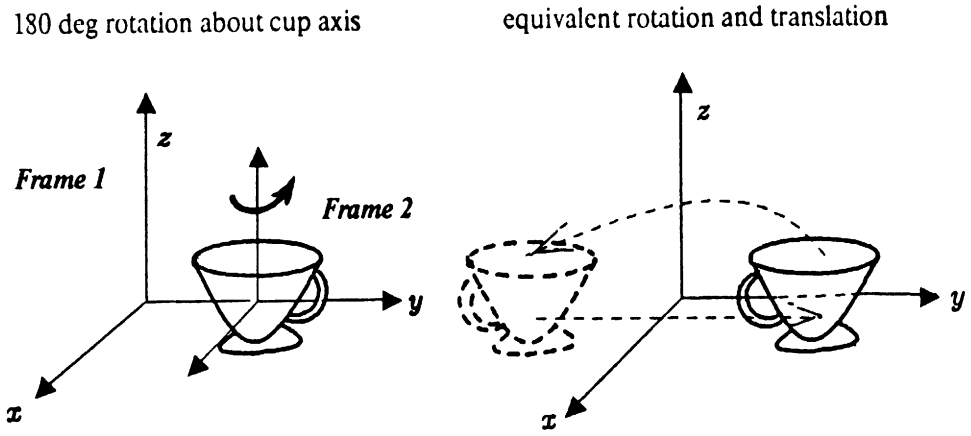



Figure 5.3: Effects of different compliance frames

and the joint stiffness relation

$$\underline{\tau} = -J_g^T K J_g \, \Delta\underline{\theta} + J_g^T \underline{\mathcal{F}}_b$$

can be used as before.

### 5.4.3 Object Centered Stiffness Control

Just as it is often desirable to specify motions of objects with respect to a body-fixed frame, we often wish to set the compliance frame to coincide with the body-fixed frame. For a diagonal stiffness matrix, the principal stiffness axes will correspond to the body-axes and the compliance center will coincide with the center of the body-fixed frame. As an example, consider the cup shown in Figure 5.3. The rotation about the cup axis is equivalent to a translation along the $x$ axis and a rotation about the $z$ axis in the first compliance frame. Hence, restoring forces will be exerted as well as moments. If the compliance frame coincides with the initial body frame, then the rotation will only cause a restoring moment to be exerted. The translational displacement now corresponds to the actual displacement of the body frame origin.

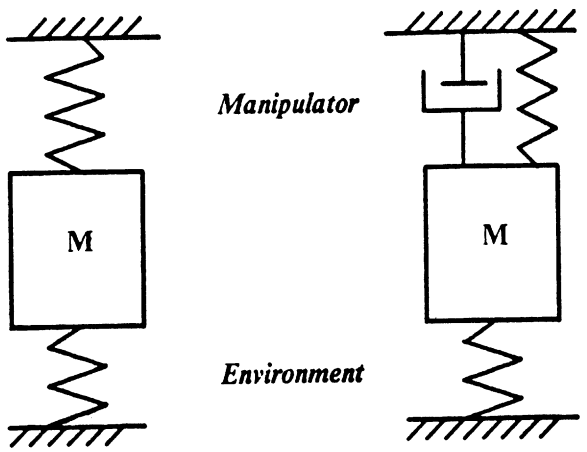



Figure 5.4: Interaction with the environment

Placing the compliance frame at a body-fixed frame $A_o$ can be easily accomplished by evaluating the grasp matrix in $A_o$. Stiffness control can then be implemented as shown previously, with the Jacobian modified to account for the frame orientation.

## 5.5 Impedance Control

If stiffness control can be perfectly implemented, the manipualted object will behave as if attached to a set of ideal springs. Consider the simple case of an object attached to a translational spring and in contact with the environment, as shown in Figure 5.4. If the environment is also modeled as a stiffness, exerting force proportional to deflection of the surface, then the closed-loop system will not be asymptotically stable. In general, excessive oscillations will result if there is insufficient damping in the environment. If the manipulator is controlled as a spring in parallel with a damper, then the closed-loop system can be made stable regardless of whether the enviroment is damped [Kazerooni 1985]. To ensure stability of the closed-loop system, we can either rely on the mechanical damping in the manipulator, i.e. imperfect implementation of stiffness control,

or introduce controlled damping. Controlled damping is preferable because it allows us to fine tune the closed-loop system to obtain the desired dynamic interactions. Relying on mechanical damping may result in an under-damped or over-damped system.

From the point of view of task stability, we need only to impart to the manipulator the characteristics of a stiffness and damper, exerting force as functions of the deviation in displacement and velocity. However, from an implementation point of view, an inertia term may also be necessary. Basically, the inertia term limits the bandwidth of the stiffness controller to reject high frequency disturbances, e.g. sensor noise and environmental vibrations. An alternate approach to the selection of impedance is suggested by Hogan [1984]. He proposed that the impedance be chosen to minimize a cost function of interaction force and motion errors, based on a-priori knowldege of the schochastic property of the uncertainties.

Impedance control of an articulated hand is a simple extension of the methodology for stiffness control. We wish to implement the linear impedance relation

$$\underline{\mathcal{F}} = -M\,\ddot{\underline{\mathcal{X}}} - B\,\Delta\dot{\underline{\mathcal{X}}} - K\,\Delta\underline{\mathcal{X}}$$

where $M$ is the inertia matrix, $B$ is the damping matrix, $\ddot{\underline{\mathcal{X}}}$ is the measured acceleration, and $\Delta\dot{\underline{\mathcal{X}}}$ is the deviation of the measured velocity from the desired nominal velocity. Using the velocity relation

$$\dot{\underline{\mathcal{X}}} = GJ\,\dot{\underline{\theta}}$$

for small displacements $\Delta\underline{\mathcal{X}}$, we have

$$\underline{\mathcal{F}} = -MGJ\,\ddot{\underline{\theta}} - BGJ\,\Delta\dot{\underline{\theta}} - KGJ\,\Delta\underline{\theta}$$

where $\ddot{\underline{\theta}}$ is the measured joint accerleration, and $\Delta\dot{\underline{\theta}}$ is the deviation in joint velocity. Using the force relation

$$\underline{\tau} = J^T G^T\,\underline{\mathcal{F}}$$

we obtain the joint impedance relation

$$\underline{\tau} = -J^T G^T MGJ\,\ddot{\underline{\theta}} - J^T G^T BGJ\,\Delta\dot{\underline{\theta}} - J^T G^T KGJ\,\Delta\underline{\theta}\ .$$

Hence

$$\underline{\tau} = -M_\theta \, \ddot{\underline{\theta}} - B_\theta \, \Delta\dot{\underline{\theta}} - K_\theta \, \Delta\underline{\theta}$$

with the joint inertia, damping, and stiffness matrices defined as

$$M_\theta = J_g^T M J_g$$

$$B_\theta = J_g^T B J_g$$

$$K_\theta = J_g^T K J_g$$

where $J_g$ is the grasp Jacobian. Again, if the compliance frame does not have the same orientation as the absolute frame, then the Jacobian need to be modified accordingly.

The material in this section is presented to show how impedance control of an articulated hand can be obtained as a simple extension of stiffness control. Implementation of impedance control requires joint velocity and acceleration sensors in addition to the position encoders for stiffness control. Real time processing of this amount of information for control of an articulated hand requires data acquisition speeds which are not achievable by present hardware. At present we have only implemented stiffness control for the Stanford/JPL Hand at the MIT Artificial Intelligence Lab. The mechanical damping of the fingers plus a fixed nominal damping in the control system was sufficient to maintain task stability. In the following chapters, only stiffness control will be used in the discussion of compliant motion of objects.

# Chapter 6

# Compliant Motion of Objects

## 6.1 Introduction

Compliant motion is in essence motion guided by the geometric constraints in the environment. By allowing geometric constaints to guide the motion of a manipulator, tasks can be successfully completed in the absence of precise information of the environment. We have seen that manipulation with an articulated hand requires not only the net motion of the object be guided by the geometry of the environment, but also that the internal grasp motions be guided by the geometry of the object.

Force control strategies are basically specifications of *how* motions are to be guided by the geometric constraints. Using the stiffness approach, restoring forces are exerted on the object proportional to the displacement from a desired nominal position. Generating compliant motion then involves superimposing stiffness control on a desired nominal trajectory. During the motion, restoring forces are exerted proportional to the deviations from the planned trajectory. In this chapter, we will integrate the trajectory computation and stiffness control methodologies developed in previous discussions. Algorithms are presented for generating compliant motion of objects. We will also study the effects of a compliance frame fixed with respect to the hand versus one fixed with respect to the object. Consideration is also given to the variations in computational

complexity for motions controlled with respect to the two classes of compliance frames.

## 6.2 Compliant Trajectory

Combining trajectory computation with stiffness control, we now have the necessary tools for generating compliant motion of objects. A joint trajectory is first computed based on the desired object motion, as shown in Chapter 4. Let $\underline{x}_d(t)$ be the contact motion which will result in the desired object motion, then the corresponding joint position at time $t_k$ is

$$\underline{\theta}_d(t_k) = \Lambda_h^{-1}\, \underline{x}_d(t_k) \ .$$

This joint position is then used as the nominal position for the joint stiffness controller derived in Chapter 5. At time $t_k$, the joint force/displacement relation is

$$\underline{\tau} = -J_g^T(t_k) K J_g(t_k)\, \Delta\underline{\theta}(t_k) \ + \ J_g^T(t_k)\, \underline{\mathcal{F}}_b$$

where $\Delta\underline{\theta}(t_k)$ is the deviation of the measured joint position from the desired position

$$\Delta\underline{\theta}(t_k) = \underline{\theta}(t_k) - \underline{\theta}_d(t_k)$$

and $J_g(t_k)$ is the grasp Jacobian defined by

$$J_g(t_k) = G(t_k)\mathcal{R}_o^T(t_k) J(t_k) \ .$$

The grasp matrix $G(t_k)$ is computed from the nominal position of the contacts $\underline{x}_d(t_k)$ relative to the compliance frame. The matrix $\mathcal{R}_o(t_k)$ is a generalized transformation matrix which transforms the contact force vectors in the compliance frame into force vectors in the absolute frame. Hence, it represents the orientation of the compliance frame at time $t_k$. The matrix $J(t_k)$ corresponds to the Jacobian at the nominal joint position $\underline{\theta}_d(t_k)$. The small displacement assumption has allowed us to transform the Cartesian stiffness relation into a pre-computable joint stiffness relation. In essence, we have linearized the

force/displacement relation in joint space about the nominal position. We will write the linearized relation at time $t_k$ as

$$\underline{\tau} = -K_\theta(t_k)\,\Delta\underline{\theta}(t_k) + \underline{\tau}_b(t_k)$$

where

$$K_\theta(t_k) = J_g^T(t_k)\,K\,J_g(t_k)$$

$$\underline{\tau}_b(t_k) = J_g^T(t_k)\,\underline{\mathcal{F}}_b \ .$$

As the trajectory $\underline{x}(t)$ must be interpolated between knot points in joint space, the stiffness control law must also be interpolated. Consider the motion between knot points $\underline{\theta}(t_k)$ and $\underline{\theta}(t_{k+1})$. A simple interpolation algorithm is to use linear interpolation of the joint positions and the goal joint stiffness and bias torque for each motion segment, i.e.

$$\underline{\tau} = -K_\theta(t_{k+1})\,\Delta\underline{\theta}(t) + \underline{\tau}_b(t_{k+1})$$

for $t_k < t \leq t_{k+1}$, where

$$\Delta\underline{\theta}(t) = \underline{\theta}(t) - \underline{\theta}_d(t)$$

with $\underline{\theta}_d(t)$ being the linearly interpolated trajectory

$$\underline{\theta}_d(t) = \underline{\theta}_d(t_k) + \frac{t - t_k}{t_{k+1} - t_k}\left[\underline{\theta}_d(t_{k+1}) - \underline{\theta}_d(t_k)\right] \ .$$

By using a constant joint stiffness and bias torque throughout the segment, this algorithm is simple and efficient. However, at the transition between segments, there is a step change in the commanded joint stiffness and bias torque. As the stiffness matrix can be viewed as the proportional gain matrix for a position control system, this is equivalent to a step change in the servo gains. Such step change in gains may cause undesirable oscillations or sudden jumps in motion. To eliminate the problems associated with the discontinuities, we may wish to interpolate the stiffness matrix and bias torque in addition to the position. A linear interpolation algorithm yields

$$\underline{\tau} = -K_\theta(t)\,\Delta\underline{\theta}(t) + \underline{\tau}_b(t)$$

for $t_k < t \leq t_{k+1}$, where

$$K_\theta(t) = K_\theta(t_k) + \frac{t - t_k}{t_{k+1} - t_k}[K_\theta(t_{k+1}) - K_\theta(t_k)]$$

$$\underline{\tau}_b(t) = \underline{\tau}_b(t_k) + \frac{t - t_k}{t_{k+1} - t_k}[\underline{\tau}_b(t_{k+1}) - \underline{\tau}_b(t_k)] \ .$$

Significant improvements in stiffness and force transition can be obtained from this slight refinement of the interpolation algorithm. However, just as a rotation matrix cannot be interpolated element by element to represent intermediate rotations, neither can the stiffness matrix be interpolated element by element to represent intermediate stiffnesses. The desired stiffness behavior can be approximated only if the knot points are closely spaced.

In generating manipulator Cartesian motion, often quadratic interpolation of joint position is used to smooth out the transition between segments [Taylor 1979]. Similary, we may wish to use quadratic interpolation for the finger joint positions as well as the stiffness matrix and bias torque. Quadratic interpolation of joint position and bias torque is feasible with current computation speeds. However, the computations required to quadratically interpolate a joint stiffness matrix is subtantial. For example, for the 9×9 stiffness matrix of the Stanford/JPL Hand, 81 elements must be interpolated. The controller servo rate must be slowed to accomodate the increase in computation time. In general, there is always a trade-off between smoothness of motion and controller bandwidth.

## 6.3 Selecting the Compliance Frame

The compliance frame can be chosen completely independent of the desired trajectory of the object. The role of the compliance frame is to define the displacement and forces. As an example, consider the cup shown in Figure 6.1. The displacement of the cup from position $A$ to $B$ is a pure rotation about the $z$ axis in the first compliance frame. Using a diagonal stiffness matrix, a pure restoring torque will be exerted about the $z$ axis of the frame. The same

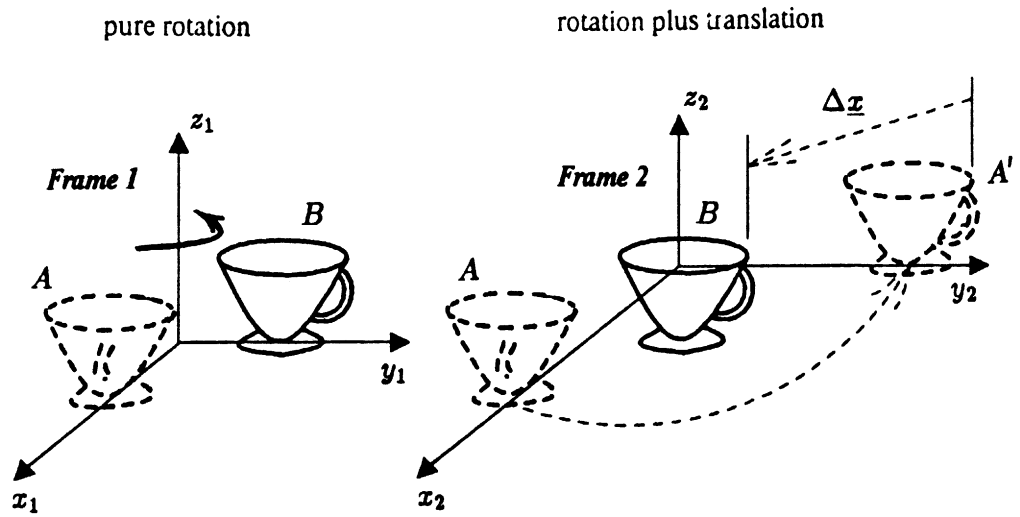



Figure 6.1: Definition of displacement in a compliance frame

displacement in the second compliance frame is obtained by a rotation from $A$ to $A'$ followed by a translation along the $\Delta x$ vector. Therefore, a net force in the $-\Delta x$ direction will be exerted as well as a moment about the $z$ axis. We see that the magnitude of translational displacement is dependent on the choice of compliance frame. The rotational displacement is a measure of the changes in the orientation of the object, and hence the magnitude is uniform in all compliance frames. Now consider force exertion in different compliance frames. Given the same contact forces, the magnitude of the net translational force on the object is uniform in all frames. However, the net moment exerted on the object is dependent on the location of the contact points relative to the frame origin. Therefore, the compliance frame is essentially a definition of the translational displacement of an object and the rotational force on the object. It should be chosen to reflect the *natural* definitions of translational displacement and rotational force for an object. For example, consider the block shown in Figure 6.2. The coordinate frame with origin at the centroid of the block is a natural choice of the compliance frame. The translational displacement is then defined as the displacement of the centroid, and the restoring moments are defined to be moments about the nominal centroid.

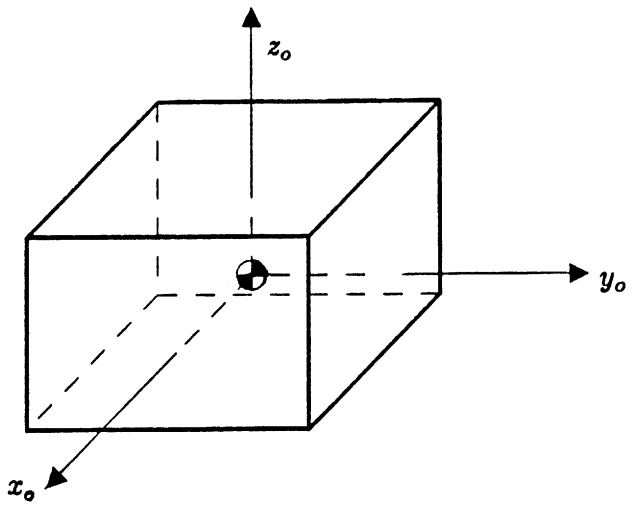



Figure 6.2: Compliance frame chosen to reflect actual translation

In most cases, the natural choice of a compliance frame is the body-fixed frame which defines the position and orientation of an object. The translational displacements in the compliance frame then correspond to the actual displacements of the object position. Restoring moments will also be exerted about the nominal "center" of the object. Coinciding the axes of the compliance frame with those of the object also simplifies the selection of the stiffness matrix. Therefore, it is usually desirable to use a compliance frame which coincides with the nominal body-fixed frame, i.e. the frame describing the nominal position and orientation of the object. As the object trajectory is executed, the compliance frame should be constantly updated to coincide with the current nominal body-fixed frame. If the compliance frame is not updated, then future translational displacements measured in the compliance frame will not represent the actual deviations in the object position. Given a rotational displacement, the restoring moment about the object center will vary as the distance of the object from the compliance center changes.

For some tasks, using a body-fixed frame as the compliance frame may not be the best choice. Analysis have shown that the ideal location of the compliance

center for peg-in-hole insertions is at the mouth of the hole [Whitney 1982]. By fixing the compliance center with respect to the hole, the risk of jamming is greatly reduced. Therefore, if the hand is stationary while the fingers perform the peg insertion, the compliance frame should be fixed with respect to the hand.

## 6.4 Computational Complexity

In this section we will compare the computational requirements for motions controlled with respect to a body-fixed compliance frame and a hand-fixed compliance frame. When implementing the joint stiffness relation

$$\underline{\tau} = -J_g^T K J_g \, \Delta\underline{\theta} + J_g^T \underline{\mathcal{F}}_b$$

the bulk of computation is contained in the evalutaion of the Grasp Jacobian $J_g$ at each knot point, where

$$J_g = G\mathcal{R}_o^T J$$

The matrix $G$ is a function only of the positions of the contacts in the compliance frame. Assuming that the fingers do not slip and that rolling at the finger tips is negligible, the positions of the contacts will remain constant relative to a frame fixed to the grasped object. Hence, for motions controlled with respect to a body-fixed compliance frame, the matrix $G$ needs only be evaluated once, i.e. the same matrix applies to all knot points. However, the orientation of the compliance frame will vary as the orientation of the object. Therefore, the orientation matrix $\mathcal{R}_o$ must be evaluated for each knot point.

Conversely, for a compliance frame fixed with respect to the hand, the positions of the contacts relative to the frame will change as the object motion proceeds. Therefore, the matrix $G$ must be evaluated for each knot point, while the rotation matrix $\mathcal{R}_o$ remains constant. The Jacobian $J$ is a function only of the joint position, and hence need to be evaluated for each knot point regardless of the compliance frame.

In general, it is simpler to evaluate a rotation matrix $\mathcal{R}_o$ than a grasp matrix $G$. Evaluation of $\mathcal{R}_o$ for a body-fixed compliance frame requires computing the

grasp frame $A_g$ from the contact positions. Let $R_g$ be the rotation matrix in $A_g$, and let $R_{o'}$ denote the orientation of the compliance frame relative to the grasp frame, then the orientation of the compliance frame is given by the rotation matrix

$$R_o = R_g R_{o'}$$

The generalized rotation matrix $\mathcal{R}_o$ can then be easily constructed from $R_o$. Evaluation of the grasp matrix $G$ requires finding the contact positions in the compliance frame, constructing $G^{-T}$ from the positions, and inverting the $G^{-T}$ matrix. For a large $G^{-T}$ matrix (9×9 for the Stanford/JPL Hand), inversion will require substantial computation time. Hence, from a computational point of view, using a body-fixed compliance frame is preferable to using one fixed with respect to the hand.

# Chapter 7

# A Hand Control Language

## 7.1 Introduction

The fundamental distinction between robots and fixed automation is programmability. Through programming, the robot can adapt to different tasks without re-design of its physical configuration. Over the years, a number of programming languages have been developed specifically for the control of robotic manipulators. The earliest of such endeavors consists of manually moving the manipulator to a desired configuration and recording the corresponding joint positions. A program is then composed of a series of joint position commands plus signals for the end effector. This *teach-by-showing* or *guiding* approach to programming is still widely used in industrial manipulators. The drawback of such methods is that motion cannot be altered via sensory feedback. The manipulator simply executes the sequence of moves as taught, without condition monitoring. This is adequate for tasks in a predictable environment, such as spray painting and spot welding. Tasks in which subsequent moves must be based on current sensory data, such as assembly and parts inspection, requires manipulator languages with data accessing and conditional branching capabilities. Some programing languages provide extensions to guiding which include testing of external binary signals and conditional branching, e.g. the ASEA [ASEA] and Cincinatti Milacron [Holt 1977] systems. In these languages, the sequence of motions taught

by guiding are each given numbers. The manipulator can branch to appropriate points in the sequence based on the conditional tests.

Some manipulator programming languages provide capabilities comparable to general purpose computer programming languages. The first of such is the WAVE system [Paul 1977] developed at Stanford. This system provided pioneering features such as the specification of manipulator position in terms of end effector Cartesian position and orientation. It also provided algorithms for smooth Cartesian trajectory segment transition and for specification of Cartesian forces. WAVE ran off-line on a machine which produced a trajectory file to be executed by another machine responsible for real-time control. This is primarily due to the time consuming sophisticated trajectory planning and inverse dynamic computations provided by the system, requiring the trajectories and forces to be pre-computed in joint space. The alogrithms were based on the assumption that the deviation from the desired path is small.

The MINI system [Silver 1973] developed at MIT was based on an existing LISP system. In essence, it consisted of a set of functions in LISP which performed the tasks of setting position and force goals and communicating with another machine which controlled the manipulator in real time. The advantages of this system is that it can be easily expanded by writing additional LISP functions. The LISP system also provides an interactive environment for immediate excution of statements and program debugging.

The AL language was motivated by the desire to develop a complete robot programming language which includes all the features specific to manipulators as well as those of a general purpose high-level language. As an extension of the ALGOL language, it has the same block structure for program control. The AL system provided all the capabilities of WAVE as well as coordination of parallel processes. Special geometric data types are defined, such as vectors, rotations, and coordinate frames. Arithmetic operators for these data types are also defined, e.g. vector products, composition of transformations, and coordinate mappings. AL provides an AFFIX statement which models the relationship between two attached frames. Whenever one of them is changed, the other will

be updated to maintain the fixed relationship. The AL system consisted of a compiler which translated the program into low level commands interpreted by a machine dedicated to real time control. Recent developments have made AL an increasingly interactive system [Goldman 1982], supporting immediate execution of single statements, setting of breakpoints, and single stepping to subsequent statements in a program.

Another comprehensive language is AML [Taylor, Summers, and Meyer 1982], used in IBM robots. Similar to the MINI system, it is designed to provide the user an environment to build other programming interfaces, e.g. vision. Like the AL language, geometric data types and operators are defined. Cartesian motion planning and affixment of frames are also supported. However, no mechanisms are provided for parallel process control and general compliant motion.

Although there exist a number of manipulator programming languages, there is to date no language designed specifically for articulated hands. The desire for high-level control of the Stanford/JPL Hand motivated the design of a hand programming language. The goal is to provide coordination of finger motions based on high-level specification of desired motion of grasped objects. The design philosophy is similar to that of AML, i.e. to provide an easily expandable vocabulary and all the basic features specific to hand programming. Since manipulator programming is a highly interactive task, requiring repeated trials, the language should be interpreted, so as to bypass the traditional edit-compile-test loop. A statement can then be immediately tried out and the state of the program can be examined for debugging. Rather than constructing a completely new language, we have used a LISP system as its basis, similar to the MINI system. The LISP system provides a rich interactive environment for programming and debugging. Immediate execution of single statements, setting of breakpoints, and single stepping through a program are all supported by the LISP environment. With the language written as a collection of LISP procedures, it can be expanded by simply defining more procedures. As most knowledge-based programs are written in LISP, the integration of knowledge-based systems with

manipulation is also simplified.

This chapter describes the LISP based system developed for the Stanford/JPL Hand. The control structure resembles the WAVE and AL systems in that joint postions and forces are pre-computed based on the assumption of small deviations. The desired position and force trajectories are generated by the LISP machine and sent to a VAX-11/750 real-time control machine for execution. First, a brief description will be given of the interaction between the LISP machine and the VAX. Then the basic repertoire of functions and high-level procedures will be described. We will conclude with a programming example for a peg-in-hole insertion task.

## 7.2 Control Heirarchy and Interaction

The hand programming system developed for the Stanford/JPL Hand consists of three levels of control. At the top level is a LMI LISP machine (CADR) which interprets high-level motion commands and translates them into a sequence of joint level commands. The joint level commands are then sent to a VAX-11/750 dedicated to real-time control. The VAX performs interpolations in joint space, reads sensor data, and sends new setpoint commands to an array of microprocessors controlling the servo motors. A schematic of the control heirarchy is shown in Figure 7.1. We will not discuss the control algorithms used for servoing to a desired position or torque, but will simply assume that the VAX/microprocessor control system accurately enforces the desired setpoints at each servo cycle. A detailed discussion of the control sytem is given in [Salisbury 1984a].

A basic data structure referred to as a "seg" is defined in the VAX. Each seg represents a trajectory knot point. It contains a set of joint positions, a time duration, and a pointer to the next seg. The time duration is the duration for moving from the last position in the previous seg to that in the current seg. The trajectory can be either linearly or quadratically interpolated, as both types of interpolation are supported. During pure position control, the VAX control

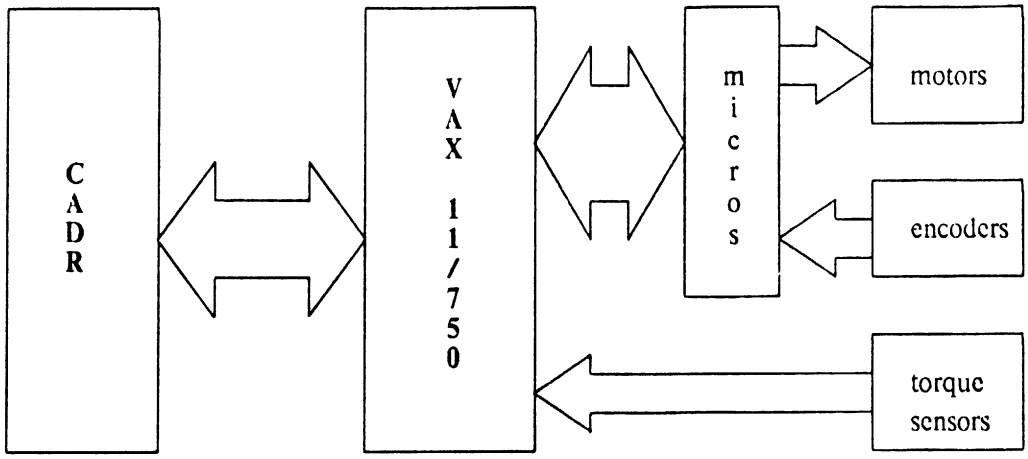



Figure 7.1: Hand programming system control heirarchy

program simply steps through a sequence of segs until a null seg is reached. To support pure force control and stiffness control, each seg also contains a set of joint torques and a pointer to a joint stiffness matrix. During pure force control, the program steps through the segs, and commands the corresponding joint torques. During stiffness control, the positions and the torques are interpreted as the nominal values. The deviations in joint positions are multiplied by the joint stiffness matrix, and the results are added to the nominal torques.[1] If a torque or a stiffness is not defined for a particular seg, then a default torque or stiffness will be used. The control program can be switched between pure position, pure force, and stiffness modes by a simple sequence of character commands.

The basic task of the LISP machine is to translate high-level motion commands into a sequence of corresponding seg structures. The LISP machine communicates with the VAX via message passing, using two DR11-C parallel interface boards. First, the LISP machine specifies whether a seg structure is to be executed by the VAX as soon as it is completed, or that it should wait for an entire sequence to be transmitted before execution. The contents of a seg is

[1] At present, torque and stiffness interpolation are not supported.

then sent, and is stored into a seg structure allocated by the VAX. The pointer to the next seg is set to that of the next structure to be allocated. When the entire sequence of segs has been specified, the LISP machine sends a begin signal and the motion is executed. The VAX deallocates the seg structures as they are freed. During the motion, additional segs may be inserted, and the VAX may send messages reporting its position or force status. As the LISP machine computes the next sequence of motions, the new sequence can be modified based on the data.

There are approximately twenty different messages which can be passed from the LISP machine to the VAX. Among these are commands to pause the VAX servo control program, change the control mode (position, force, or stiffness), add a seg, set torque, set stiffness, set a conditional, start trajectory, query position, query torque, and query finger forces. The finger forces correspond to those which will be obtained from recently developed finger tip sensors [Salisbury 1984b]. Also included are commands to define a position, torque, or stiffness by a character string. Definition by a character string is used when a position, torque, or stiffness matrix is used repeatedly. A short character message can then replace a long sequence of floating point numbers. The efficiency in message passing appears to merit the extra effort spent in contructing a symbol table for the VAX.

There are seven different messages which can be passed from the VAX to the LISP machine. Most of these messages are status reports, including current position, torque, finger forces, and general status. The remaining are print character string, and general integer and floating point data responses to query. This set of messages is sufficiently general to report any situations the VAX may encounter. As experience is gained with using the system, more specialized responses will be defined. A listing of the messages and syntaxes is given in Appendix A.

## 7.3 Basic Functions

The hand programming system developed is basically an extension to an existing LISP system. LISP procedures are defined for computations specific to the hand and for communicating with the VAX via parallel connections. The LISP procedures can be divided into two levels — a functional level and a higher *object-oriented* programming level. The first consists of a set of basic functions to perform coordinate frame and kinematic transformations, compute grasp frame and grasp matrix, generate trajectory of Cartesian points, and compose message packets. These are functions called by names and arguments, as in programming languages such as Fortran and C. The next level of procedures are written in the *message passing* style of programming, also referred to as *object-oriented programming*. This style of programming is used in the Smalltalk and Actor families of languages, and is also supported by the LISP system. When a particular action is desired, a message is sent to an abstract object which performs the required operations. Each abstract object is basically a data structure which can be accessed and modified according to the messages received. Because each object is a data structure, the results of previous computations and internal states can be retained. This can free the programmer from tedious bookkeeping and simplify the program. Another advantage of object oriented programming is modularity. Each object is self-contained and presents the caller with a set of external interfaces, i.e. the defined messages. The caller is not required to understand the implementation details, but only that a particular message sent to an object will cause a particular action to be performed.

The functional procedures will be described in this section. The object-oriented procedures are built on these basic functions, and will be described in the subsequent sections. This is not meant to be a comprehensive documentation, but only a description of the basic features provided in the hand programming system. The design philosophy is to adopt existing manipulator language syntax when possible, and to make the arguments optional when possible. By adopting existing language syntax, users familiar with other manipulator languages can easily adapt to the new system. By making most arguments

optional, a new user can quickly begin programming without having to learn detailed function syntax. As experience is gained, the user can progressively use the optional arguments to exercise more control over the program. Only the optional arguments pertinent to our development of a simple working program will be described here. The codes are written in the Zetalisp dialect. It is assumed that the reader has some familiarity with the list evaluation structure of LISP;[2] the special features provided by Zetalisp will be explained as necessary.

### 7.3.1 Frame Representation

Many of the ideas for coordinate frame representation were taken from the AL language. Rotations are represented by 3×3 matrices. A rotation, or *rot*, can be constructed from a unit vector, specifying the axis of rotation, and a scalar, specifying the angle of rotation. The function `make-rot` returns a rot, and is called by the following syntax

```
(make-rot rot-vector :thru angle)
```

The argument `rot-vector` is a one-dimensional array of 3 elements, and `angle` is given in radians. The argument `angle` is referred to as a *key-worded* argument. Key-worded arguments are optional arguments which are associated by key-words with the appropriate variables in the function. Hence, the key-word `:thru` associates the subsequent argument with an angle. If the optional argument is omitted, then `vector` will be interpreted as the set of Euler angles describing the rotaton. Hence, `(make-rot rot-vector)` is also a valid function call. As is, a new 3×3 array is allocated and returned every time the function is called. To minimize the work of the garbage collector, the results can be stored into an existing 3×3 array by providing another key-worded argument

```
(make-rot rot-vector :thru angle :into storage-array)
```

The `:into` option is available in all functions which return a vector or an array.

Computations of rotation matrices are greatly simplified when the axis of

[2] See [Winston and Horn 1981] for a tutorial on the LISP language.

rotation is one of the coordinate axes. The `make-rot` function also recognizes special symbols indicating rotation about a coordinate axis. Greater efficiency is obtained by using these symbols instead of the actual unit vectors. For example, use

```
(make-rot 'xhat :thru angle)
```

instead of

```
(make-rot (vector 1.0 0.0 0.0) :thru angle) .
```

A coordinate frame is represented by a 3×4 matrix, the first three columns is the rotation matrix, and the last column is a 3-element vector specifying the displacement of the origin. A coordinate frame, or simply *frame*, is constructed from a rot and a displacement vector by using the `make-frame` function

```
(make-frame rot vector) .
```

The inverse of a frame is returned by evaluating

```
(invert-frame frame) .
```

Composite transformations are obtained by multiplying the frames. Because the frames are represented internally as 3×4 arrays, a special frame multiplication function is provided. Frames are multiplied in succession by using

```
(multiply-frames frame-1 frame-2 frame-3 ... ) .
```

A new frame is returned. There is no limit to the allowable number of arguments.

### 7.3.2 Transforming Points and Vectors

Transforming points expressed in one frame into those in another frame is done by the `transform-points` function. The syntax is similar to that used by AML, but offers much more flexibility. Positions expressed in a realtive frame can be transformed into positions expressed in the "absolute" frame by

```
(transform-points frame-foo pos-array) .
```

The argument `pos-array` is a one or two-dimensional array containing the coordinates of points expressed in `frame-foo`. Each row contains the x-y-z coordinates of an arbitrary number of points. For example, a typical row is

$$\begin{bmatrix} x_1 & y_1 & z_1 & x_2 & y_2 & z_2 & \cdots \end{bmatrix}$$

This function basically multiplies three coordinates at a time by the frame matrix. An array of the same dimensions as `pos-array` is returned, containing the coordinates of these points expressed in the "absolute" frame. The ability to transform coordinates contained in two dimensional arrays is convenient for transforming entire trajectories of finger positions. Similarly, the `transform-vectors` function re-expresses vector directions, e.g.

```
(transform-vectors frame-foo vect-array) .
```

Here `frame-foo` can be either a frame or simply the corresponding rotation matrix; in either case, a vector is only rotated by this function and its length is preserved.

### 7.3.3 Hand Kinematic Transformations

The kinematic transformation $\Lambda_h(\underline{\theta})$ from joint space to Cartesian space is performed by the function

```
(j-to-c joint-pos)
```

The argument `joint-pos` is a one or two-dimensional array containing the joint positions. Each row contains nine elements, with the positions of finger 1 occupying the first three elements, those of finger 2 occupying the next three elements, etc. Hence, each row corresponds to a complete hand configuration in joint space. An array of the same dimensions is returned containing the corresponding finger tip positions. The first three elements in each row contain the x-y-z coordinates of finger 1, the next three elements contain those of finger 2, etc.

The inverse kinematic transformation $\Lambda_h^{-1}(\underline{x})$ from Cartesian to joint space is performed by the function

```
(c-to-j cartesian-pos)
```

The argument `cartesian-pos` is a one or two-dimensional array containing the positions of the finger tips. Each row contains nine elements, with the position of each finger occupying three consecutive elements. Therefore,

```
(j-to-c (c-to-j cartesian-pos))
```

will return a copy of `cartesian-pos`. Recall that in general two solutions are possible for the inverse kinematic transformation of each finger. By default, the solution selected corresponds to all fingers curling in toward the palm. Alternate solutions are obtained by including a key-worded argument as follows:

```
(c-to-j cartesian-pos :curl-out curl-vector) .
```

The argument `curl-vector` is a three-element vector of logical values, i.e. T or NIL. A non-NIL value for an element indicates that the corresponding finger is to curl outward from the palm. For example, evaluation of

```
(c-to-j cartesian-pos :curl-out (vector t nil nil))
```

will return the set of solutions corresponding to finger 1 curling outward from the palm and fingers 2 and 3 curling in toward the palm. In the event that a certain hand configuration connot be attained, the function will return a list containing two integers. The first integer is the row number in the `cartesian-pos` array at which this occured; the second integer is an error code indicating the type of error. There are four possible types of error for each finger. The first three correspond to a required joint angle being greater than the limits imposed by the mechanical design. The fourth is that the desired position is simply out of reach, regardless of joint limits. The first four bits of the error code represent these errors occuring in finger 1, the next four bits represent these errors in finger 2, etc. Therefore, given a desired Cartesian trajectory, the returned list can indicate the point in the trajectory at which an error will occur, as well as the type of error.

The transpose of the Jacobian matrix is found by evaluating

```
(jac-transpose joint-pos)
```

The argument `joint-pos` is a vector of nine elements, containing the finger joint positions. This function returns a 3×9 "compressed" Jacobian transpose, with the 3×3 Jacobian transpose of each finger occupying three consecutive columns. A function was provided for evaluating the Jacobian transpose rather than the Jacobian itself because the transpose is used more often in practice.

Corresponding transform functions for individual fingers are also provided. They have names such as `j-to-c-1`, `j-to-c-2`, etc. Since we are only concerned with coordinated motion of the entire hand, these functions will not be discussed here.

### 7.3.4 Grasp Frame and Grasp Matrix

As described in Section 4.3, the grasp frame for the Stanford/JPL Hand is defined by the triangle formed by the grasp points. A 3×4 matrix corresponding to the grasp frame is returned by evaluating

```
(make-grasp-frame cartesian-pos)
```

where `cartesian-pos` is a vector of nine elements containing the positions of the finger tips. By default, the origin of the grasp frame is located at the centroid of the grasp triangle. The origin can also be set at one of the grasp points by including a key-worded symbol. For example,

```
(make-grasp-frame cartesian-pos :origin 'fing1) .
```

will place the origin at the tip of finger 1.

As with the Jacobian, the transpose of the grasp matrix is evaluated instead of the grasp matrix itself. A 9×9 grasp matrix transpose (see Section 5.3) is returned by

```
(grasp-matrix-transpose cartesian-pos) .
```

The argument `cartesian-pos` is a vector of nine elements containing the positions of the finger tips relative to the frame in which forces and moments are

defined, e.g. the compliance frame. The transpose of the grasp Jacobian is constructed by evaluating

```
(grasp-jac-transpose jac-transpose frame grasp-mat-transpose) .
```

The first argument is the 3×9 compressed Jacobian transpose. The second argument is the 3×4 compliance frame, or simply a 3×3 rot describing the orientation of the compliance frame. The third argument is the transpose of the grasp matrix. This function is basically an efficient algorithm for constructing the grasp Jacobian transpose defined by

$$J_g^T = \begin{bmatrix} J_1^T & 0 & 0 \\ 0 & J_2^T & 0 \\ 0 & 0 & J_3^T \end{bmatrix} \begin{bmatrix} R_o & 0 & 0 \\ 0 & R_o & 0 \\ 0 & 0 & R_o \end{bmatrix} G^T$$

where $J_i$ is the $i^{th}$ finger Jacobian and $R_o$ is the 3×3 rotation matrix of the compliance frame.

### 7.3.5 Generating a Trajectory

Trajectories of points in Cartesian space are generated by the function

```
(gen-cartesian-traj cartesian-pos move-spec-list :nseg nseg) .
```

The first argument is a vector of arbitrary length containing the x-y-z coordinates of the points to be moved, with the coordinates of each point occupying three consecutive elements. The second element is a list of *move-specs*. There should be as many move-specs as there are points in `cartesian-pos`. Each move-spec specifies the motion of the corresponding point. If there is only one move-spec in the list, then that move will apply to all the points. A move-spec is itself a list of the form

```
(rot-vector angle trans-vector)
```

where `rot-vector` is a unit vector about which the point will be rotated through `angle`, and `trans-vector` is a three-element vector specifying the translation in the x-y-z directions. The resulting motion is a translation superimposed

on a rotation. The key-worded argument nseg is the number of knot points to compute for the trajectory, uniformly distributed in time. For $n$ points, cartesian-pos will be a vector of length $3n$, and the trajectory will be returned as a nseg$\times 3n$ array. If nseg is omitted, the value defaults to 1, i.e. only the final positions are computed.

Frequently, a pure rotation or a pure translation is desired, or a particular point does not need to be moved. In these cases, the following forms of move-spec will make computation more efficient:

| | | |
|---|---|---|
| (NIL NIL NIL) | ⇒ | null move, the point will remain at the orginal coordinates |
| (rot-vector angle NIL) | ⇒ | pure rotation |
| ('xhat angle NIL) | ⇒ | pure rotation |
| (NIL NIL trans-vector) | ⇒ | pure translation |

### 7.3.6 Sending a Trajectory

As described previously, the basic task of the LISP machine is to fill a data structure in the VAX, referred to as a "seg". This is accomplished by sending the VAX message *packets* which contain operation codes, or op-codes, and data. The packets are transmitted via a 16-bit parallel interface, and hence data must be converted into sequences of 16-bits. The function which performs this task is called by the following syntax:

```
(make-packet op-code1 op-code2 data1 data2 data3 ... ) .
```

The first two arguments are integer op-codes which define the operations to be perfomed. The remaining arguments are data which are expected to follow the op-codes. They can be character strings, single integers, single floating point numbers, or arrays of numbers. The make-packet function will convert all character strings into pairs of 8-bit ASCII codes, adding a null padding character if needed to complete a 16-bit word. All integers are converted into 16-bit "short integer" formats, and all floating point numbers are converted into 32-bit formats used by the VAX. This function returns a vector of 16-bit

elements which can be sent through the parallel interface an element at a time. Each packet also has a header and a tail containing information such as packet i.d. number and the number of words to be transmitted.

As an example of the use of this function, we will compose a packet containing a message to move the fingers to a specified position. To command a motion of the fingers, the following packet is sent:

```
(make-packet add-seg 0 joint-pos duration)
```

where `add-seg` is a pre-defined integer op-code for adding a trajectory segment. The second op-code is currently not used, and hence can be any integer. The argument `joint-pos` is a vector specifying the joint positions in radians, and `duration` is the time duration in seconds for this move.

Joint stiffness matrices are stored into the VAX seg structures by sending a `set-stf` message. A stiffness matrix is always sent before the corresponding joint positions. When a `add-seg` message is received, a seg is considered completely specified. A default stiffness will be used if `add-seg` was not preceeded by a `set-stf` message. The same rules apply to storing torque vectors into seg structures. The op-code definitions and packet syntaxes are given in Appendix A. The mechanism for sending message packets is written in the object-oriented programming style, and hence its description will be deferred to the next section.

### 7.3.7 An Example

As an illustration of their uses, we will generate a trajectory using these basic functions. Assume that the body-fixed frame of a grasped object has the same orientation as the grasp frame, but its origin is translated along the $y$-axis of the grasp frame by 2 centimeters (see Figure 7.2). We wish to rotate the object about its own $x$-axis by 45 degrees.

Let the current joint positions be contained in the 9-element vector `joint-pos`. We first evaluate the current Cartesian positions and grasp frame by

```
(setq cartes-pos (j-to-c joint-pos))
```

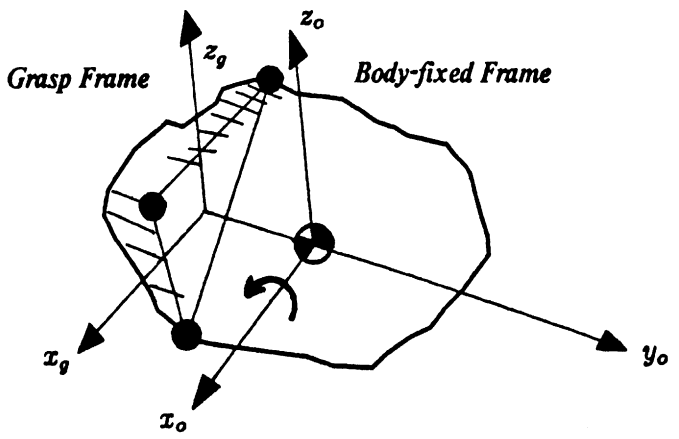



Figure 7.2: Frame definition for programming example

```
(setq grasp-fm (make-grasp-frame cartes-pos)) .
```

Next, the frame describing the object relative to the grasp frame is obtained.

```
(setq rel-frame (make-frame NIL (vector 0.0 2.0 0.0)))
```

The current body-fixed frame is then computed from

```
(setq body-fm (multiply-frames grasp-fm rel-fm)) .
```

To generate a trajectory with respect to any frame, the grasp points must first be expressed relative to the frame.

```
(setq rel-pos (transform-points (invert-frame body-fm) cartes-pos))
```

A Cartesian trajectory is then generated with respect to the body-fixed frame using a move-spec-list containing only one move-spec, i.e.[3]

```
(setq spec-list (list (list 'xhat (// pi 4.0) NIL)))
(setq cartes-traj (gen-cartesian-traj rel-pos spec-list :nseg 10))
```

for a trajectory with 10 knot points. The variable `cartes-traj` is now bound to

[3]The Zetalisp symbol for division is the double slash // .

a 10×9 array. The trajectory expressed in the "absolute" hand frame is obtained from

```
(transform-points body-frame cartes-traj :into cartes-traj)
```

with the results stored back into the `cartes-traj` array. The Cartesian trajectory can then be transformed into a joint trajectory.

```
(setq joint-traj (c-to-j cartes-traj))
```

The variable `joint-traj` is now bound to a 10×9 array, with each row containing a configuration in joint space. Each row of the trajectory can then be sent to the VAX via the `add-seg` message.

Suppose we wish to use the nominal body-fixed frame as the compliance frame. The joint stiffness matrix corresponding to the initial position is evaluated by the following sequence of instructions:

```
(setq jac-t (jac-transpose joint-pos))
(setq grasp-mat-t (grasp-mat-transpose rel-pos))
(setq grasp-jac-t (grasp-jac-transpose jac-t body-fm grasp-mat-t))
(setq grasp-jac (transpose-matrix grasp-jac-t))
(setq joint-stf (multiply-matrices grasp-jac-t cartes-stf grasp-jac))
```

where `cartes-stf` is a 9×9 Cartesian stiffness matrix.

As can be seen from this example, computing trajectories and stiffnesses requires tedious bookkeeping of transformations and various matrices. The task of programming is greatly simplified by using the object-oriented procedures described in the subsequent sections.

## 7.4 Object-oriented Programming

Programming the hand can be simplified and made more "natural" by building a higher level interface between the programmer and the basic functions. The tedium of keeping track of positions, frames, transformations, and constructing and sending message packets should be hidden from the programmer. This

is achieved by providing an object-oriented programming interface. Using this interface, high-level programming of the hand is reduced to the sending of messages to two abstract objects. The first object is a *trajectory generator*, which is responsible for generating compliant trajectories and composing message packets. The second object is a *parallel connection* which is responsible for sending and receiving message packets.

The two components of the object-oriented programming interface will be described in this section. However, it is useful to begin with an introduction to the concepts of abstract objects and message passing.

### 7.4.1 Abstract Objects and Message Passing

An abstract object is basically a data structure containing state information. For example, an object named `my-ship` may contain informations such as position, heading, velocity, and a passenger list. Objects which have the same data structure format are said to be of the same *type*. Procedures can be defined which can operate on objects of the same type. For example, a procedure called `ship-position` can be defined to retreive the position information in a ship data structure. To obtain information on the position of a particular ship, the procedure must access that particular structure. In Zetalisp , this is accomplished by the following syntax

```
(send my-ship :ship-position) .
```

To access the position of another ship-type object named `her-ship`, we use

```
(send her-ship :ship-position) .
```

Although the process consists of passing an object to a defined procedure, it is convenient to think of it as *sending a message* to an object, hence the `send` syntax. The defined procedures can be viewed as operations "taught" to an object. The object "responds" to the messages sent to it by performing the appropriate operations. In the previous example, a ship responds to the `ship-position` message by returning its current position. The general syntax of a

message is

```
(send object operation arguments) .
```

This sends `object` a message to perform the specified `operation` using the `arguments` defined for the procedure. For example, an operation named `fractional-speed` can take an argument to specify the ship speed as a fraction of the maximum speed. Hence, to set the speed of `my-ship` at half the maximum speed, we send the following message

```
(send my-ship :fractional-speed 0.5) .
```

The ship can respond to this message by computing the new speed and appropriately update the velocity information in its structure.

### 7.4.2 The Trajectory Generator Object

The trajectory generator is the key component in the object-oriented programming interface. It contains facilities for storing frames, joint trajectories, stiffness matrices, torque vectors, default parameter values, and various other mechanisms for keeping track of the current states. Hence, a complete trajectory with the associated stiffnesses and bias torques can be stored in a trajectory generator. A trajectory is constructed by sending appropriate messages to the trajectory generator. When a message for executing the trajectory is received, the trajectory is translated into a sequence of message packets and sent to the VAX via the parallel connection object.

There are two types of trajectory generators defined. The first is referred to as a *basic-frame* trajectory generator, and the second is referred to as a *grasp-frame* trajectory generator. A trajectory generator of the appropriate type is created by evaluating the expression

```
(setq tg (make-instance 'trajectory-generator :type :basic))
```

or

```
(setq tg (make-instance 'trajectory-generator :type :grasp))
```

The variable tg is now bound to a trajectory generator object. A trajectory generator contains an internal frame which can be set by sending a message of the form

```
(send tg :set-frame frame)
```

This defines the frame in which motions and compliances are specified. For a basic-frame trajectory generator, frame is taken to be relative to the "absolute" hand frame, i.e. a frame fixed with respect to the hand. For a grasp-frame trajectory generator, frame is taken to be relative to the grasp frame, i.e. a body-fixed frame. Hence, the basic-frame trajectory generator is intended for handling motions specified relative to a hand-fixed frame, and the grasp-frame trajectory generator is intended for handling motions specified relative to a body-fixed frame. To use a basic-frame type to generate body-centered motions and compliances would require the programmer to handle the computations of the grasp frame and the body-fixed frame. Each time a motion is desired, a new set-frame message must be sent to update the internal frame. By constrast, the grasp-frame type automatically updates the grasp frame and computes the current body-fixed frame for each motion specification. When a trajectory generator is created, this internal frame is initially set to be an identity frame. Hence, if the set-frame message is not sent, the basic-frame trajectory generator will interpret the desired motions as specified in the hand frame, and the grasp-frame trajectory generator will interpret them as specified in the current grasp frame.

A grasp-frame trajectory generator can be made to imitate a basic-frame type by sending the message

```
(send tg :back-to-basic) .
```

The frame which was defined by the set-frame message is then taken to be relative to the absolute hand frame. When a back-to-basic message is received, the trajectory generator sets its grasp frame to be an identity frame and stops updating the grasp frame. Hence, frame is effectively specified with respect to the absolute hand frame. To return the grasp-frame trajectory generator to

its normal mode of operation, an internal logical variable called auto-frame-update needs to be reset by

```
(send tg :set-auto-frame-update T)
```

Using a grasp-frame type to generate motions in a stationary frame is slightly less efficient than directly using the basic-frame type. However, the ability to switch between frame types was found to be a desirable feature, and hence the basic-frame trajectory generator is rarely used. A listing of the messages which are handled by the trajectory generator is given in Appendix B.

### 7.4.3 The Parallel Connection Object

The parallel connection object is responsible for communication with the VAX. When a parallel connection object is created, a background process is set up to handle data transfers via the DR11-C interface boards connected to the VAX. As with the trajectory generator, there are two types of parallel connections — a *polled* type and a *fast-polled* type. The two types of parallel connections differ in the rate at which they check the DR11-C interface for new data. Checking the interface at a higher rate increases the bandwidth between the two machines, but takes up more process time which can be used for computations. An interrupt-driven type parallel connection object was also attempted, but could not be successfully implemented due to hardware problems in the CADR. A parallel connection object is created by evaluating

```
(setq pc (make-instance 'parallel-connection :type :fast-polled)) .
```

The variable pc is now bound to a fast-polled parallel connection object. To send a message to the VAX, a message packet is first constructed by using the make-packet function. For example,

```
(setq msg1 (make-packet add-seg 0 joint-pos 1.0) .
```

The packet is then sent by

```
(send pc :send-pkt msg1) .
```

A similar process exists in the VAX which can send message packets to the LISP machine. To check if any packets have been received from the VAX, the following message is sent

```
(send pc :data-available) .
```

The parallel connection will respond by returning either T or NIL, indicating whether there are packets already received by the background process but not yet acknowledged by the superior process. The backgraound process basically places the packets in a queue. Each packet is removed from the queue as it acknowledged by the superior process. The packets are removed from the que by evaluating

```
(setq vax-msg (send pc :get-next-pkt) .
```

The variable vax-msg is now bound to a packet, and can be passed as an argument to a procedure which handles the messages from the VAX. A listing of the messages which are handled by the parallel connection object is given in Appendix C.

## 7.5 Constructing a Trajectory

A trajectory generator maintains an internal model of the current hand configuration. This model is initialized by the message

```
(send tg :init-pos joint-pos)
```

where joint-pos is a vector of the current joint positions. If this message is not sent, then the trajectory generator will assume that the initial position is the "home position" corresponding to all joint angles being zero.

A trajectory is constructed by sending messages to the trajectory generator specifying the desired motions. The number of knot points added to the trajectory by each motion is dependent on the type of motion. For example, moving from one joint position to another adds only one knot point, whereas rotating an object usually requires more than five knot points. The trajectory generator

responds to each motion message by returning an integer indicating the total number of knot points in the current trajectory. If a particular motion cannot be achieved, e.g. a position is out of reach, the trajectory generator will print out a statement explaining the reason and return NIL. Here we will describe the messages for constructing a trajectory. Joint stiffnesses and bias torques are superimposed on the trajectory by messages which will be described in the next section. Because in practice the grasp-frame type is used almost exclusively, we will focus on the operations supported by the grasp-frame trajectory generator.

### 7.5.1 Motions in Joint Space

Motions in joint space are specified by the message

```
(send tg :move-joints-to joint-pos) .
```

The joints will move to the location given by `joint-pos` with a default time duration of two seconds. A different time duration is obtained by including a key-worded argument. The message

```
(send tg :move-joints-to joint-pos :duration 1.0) .
```

will move the joints to `joint-pos` with a one second duration. The `:duration` option is available in all messages which specify motion. If we wish all motions to have a duration of one second, it is cumbersome to repeat the `duration` key-word and argument. The default time duration can be changed by

```
(send tg :set-default-move-duration 1.0) .
```

Then all subsequent motions specified without the duration argument will have a duration of one second.

Instead of specifying the motion in terms of goal joint positions, we may wish to specify the motion as incremental changes from the current positions. Incremental motions in joint space can be obtained with

```
(send tg :move-joints-by joint-pos) .
```

The resulting joint position will be the current position incremented by `joint-pos`.

### 7.5.2 Finger Motion

The finger tips can be moved to specified coodinates in Cartesian space by sending the message

```
(send tg :move-fingers-to cartes-pos)
```

where `cartes-pos` is a nine-element vector of the desired finger tip positions and the default time duration is used. For a basic-frame trajectory generator the positions are interpreted as specified in the frame defined by the `set-frame` message. For a grasp-frame trajectory generator, the positions are interpreted as specified in the body-fixed frame. This message is normally used to perform initial grasping, before any valid grasp frame or body-fixed frame can be defined. Hence, for a grasp-frame trajectory generator, we would send a message to first convert it into a basic-frame type, and then set the appropriate frame for specifying the motion. For example,

```
(setq identity-frame (make-frame NIL NIL))
(send tg :back-to-basic)
(send tg :set-frame identity-frame)
(send tg :move-fingers-to cartes-pos)
```

will move the fingers to `cartes-pos` in the absolute hand frame. Because moving the fingers to positions in the absolute frame occurs so often, an optional arugment is provided to perform this task without having to convert the grasp-frame trajectory generator or altering the existing frame. The previous motion can be accomplished by the single message

```
(send tg :move-fingers-to cartes-pos :in-hand-frame T)
```

Analogous to its joint space counterpart, incremental motions of the fingers can be obtained from

```
(send tg :move-fingers-by cartes-pos)
```

where `cartes-pos` is again taken to be specified in the appropriate frame.

The finger solutions, i.e. finger curl configurations, are selected based on the logical state of a three-element vector in the trajectory generator. The elements of the vector are initially set to be NIL, indicating that the solutions should correspond to the fingers curling in toward the palm. Alternate solutions are obtained by sending a `set-curl` message followed by the desired logical vector. For example,

```
(send tg :set-curl (vector T NIL NIL ))
```

will select the outward curl solution for finger 1.

### 7.5.3 Object Motion

After the hand has securely grasped an object, the desired motion of the object can be obtained by sending the message

```
(send tg :generate-traj move-spec-list)
```

with the `move-spec-list` containing only one element — the move-spec describing the desired object motion. If unspecified, the duration of the move will correspond to the current `default-move-duration`. Also by default, the trajectory will be generated with five knot points, and hence use five seg structures in the VAX. The desired number of knot points is specified by including a key-worded agument, e.g.

```
(send tg :generate-traj move-spec-list :number-of-segs 10) .
```

As with the default time duration, the default number of knot points can be set by sending the message

```
(send tg :set-default-segs-per-move 10) .
```

For a basic-frame trajectory generator, the motion is specified with respect a hand-fixed frame. For a grasp-frame trajectory generator, the motion is specified with respect to the current body-fixed frame. The grasp frame is updated and the current body-fixed frame is computed *prior* to every motion command. Frequently we wish to continue a motion specified with respect to a previous

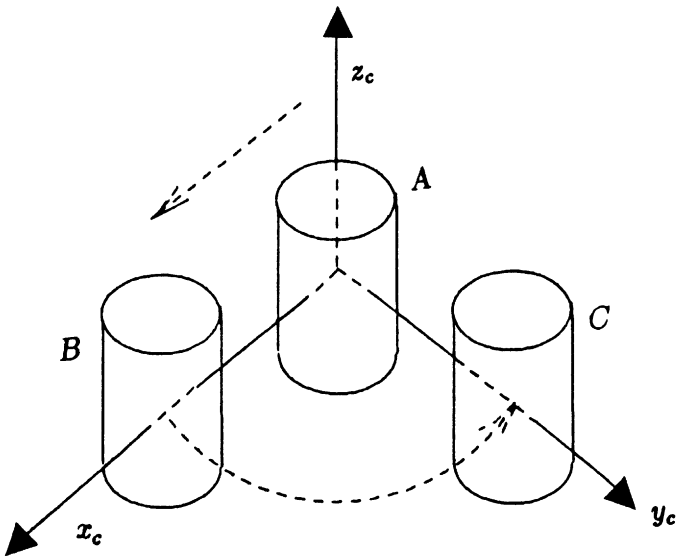



Figure 7.3: Specifying motion with respect to previous frame

body-fixed frame. For example, consider the cylinder shown in Figure 7.3. We wish to first translate the cylinder along its own $x$-axis, from position $A$ to $B$, then rotate it about the initial $z$-axis, from $B$ to $C$. In this case, we do not want the body-fixed frame to be updated for the second motion command. This sequence of motions is accomplished by

```
(setq move1 (list (list NIL NIL (vector xdist 0.0 0.0))))
(setq move2 (list (list 'zhat angle NIL)))
(send tg :generate-traj move1)
(send tg :set-auto-frame-update NIL)
(send tg :generate-traj move2) .
```

All subsequent motion commands will be interpreted as specified with respect to the initial body-fixed frame. Updating the body-fixed frame can be resumed by resetting `auto-frame-update` to T.

To specify motions with respect to a hand-fixed frame, we must convert the grasp-frame trajectory generator into a basic-frame type, as shown previously. In the special case that the motion is specified with respect to the hand frame,

the :in-hand-frame optional argument can be used, i.e.

```
(send tg :generate-traj move-spec-list :in-hand-frame T) .
```

Generating motions of objects requires only one move-spec because the same motion applies to all fingers. However, when regrasping an object or exploring its surface with a finger, different motions need to be specified for the individual fingers. This is the reason that the generate-traj message was desinged to take a list of move-specs as its argument rather than a single move-spec. This feature can be used to implement stable regrasp [Fearing 1984] and object recognition algorithms [Grimson and Lozano-Perez 1984].

### 7.5.4 Positioning and Orienting Objects

An object can be rotated about the center of the body-fixed frame to a desired orientation by

```
(send tg :orient-with frame-foo) .
```

where frame-foo is a frame or a rot specified relative to the hand frame. The object will be rotated such that its body axes will have the same orientation as the coodinate axes of frame-foo. If unspecified, the number of knot points used for the rotation is the current value of default-segs-per-move.

An object can be moved to a desired position and orientation by sending the message

```
(send tg :move-to frame-foo) .
```

The object will be simultaneously translated and rotated about its origin such that the body-fixed frame will coincide with frame-foo. Again, both duration and number of knot points can be specified or defaulted.

## 7.6 Setting Stiffness

There are in general three levels of motion specifications — motion in joint space, finger motion, and object motion. The messages for specifying these motions were described in the previous section. The trajectory generator also provides facilities for specifying stiffnesses and forces at the joint, finger, and object levels. The messages for specifying these stiffnesses and forces will be described in this section.

### 7.6.1 Joint Stiffness and Force

A trajectory is constructed from a sequence of knot points. As messages specifying motions are received by the trajectory generator, new knot points are added to the trajectory. The joint stiffness at the current knot point, i.e. the end of the current trajectory, can be specified by sending the message

```
(send tg :set-joint-stf stf-mat) .
```

where `stf-mat` is the desired joint stiffness matrix. For example, assume that a new trajectory is constructed by the sequence of messages

```
(send tg :move-joints-to joint-pos)
(send tg :set-joint-stf stf-mat1)
(send tg :generate-traj move-spec-list :number-of-segs 9)
(send tg :set-joint-stf stf-mat2) .
```

Then the joint stiffness at the first knot point will be `stf-mat1`, and that at the tenth knot point will be `stf-mat2`.

A corresponding nominal torque can be set by including the optional argument

```
(send tg :set-joint-stf stf-mat :nom-trq trq-vect) .
```

where `trq-vect` is a vector of the desired nominal bias torque.

To set the stiffness and/or torque at a preceding knot point, another optional argument is used, specifying the knot point number, e.g.

```
(send tg :set-joint-stf stf-mat :seg-number 8) .
```

The nomenclature `:seg-number` is a reminder that each knot point corresponds to a seg structure in the VAX. The first knot point is assigned the number zero, according to LISP array index convention. Hence, this message has the effect of setting the joint stiffness at the ninth knot point. If we attempt to set the stiffness at a non-existent knot point, i.e. past the end of the current trajectory, an error message will be printed and NIL is returned.

When a particular stiffness matrix is used repeatedly, sending the same long sequence of floating point numbers to the VAX is extremely inefficient. This can be avoided by associating the stiffness with a name. The VAX provides a symbol look-up table for named stiffnesses. Once the stiffness is defined, subsequent messages for using the stiffness need only contain a short character string corresponding to the associated name. These operations are handled by sending a `define-stf` message to the trajectory generator, e.g.

```
(send tg :define-stf 'foo stf-mat)
```

will associate the name `foo` with the joint stiffness matrix `stf-mat`. Setting the joint stiffness at the knot points can then be accomplished by using the symbolic name instead of the actual stiffness matrix, e.g.

```
(send tg :set-joint-stf 'foo :seg-number 8)
```

Not specifying the stiffness at a particular knot point will cause the VAX to use a default stiffness for the corresponding seg structure. This stiffness has the special name `nom-stf`, its value can be replaced by sending the message

```
(send tg :define-stf 'nom-stf stf-mat) .
```

This will cause `stf-mat` to be used at all knot points without spcified stiffness.

Frequently we may wish to only specify the nominal bias torque at a knot point and use the default stiffness. This is accomplished by

```
(send tg :set-trq trq-vect)
```

where an optional `seg-number` can also be included.

Similarly, torque vectors can be associated with a symbolic name by

```
(send tg :define-trq 'bar trq-vect)
```

Subsequent `set-trq` messages may then use the symbol `bar` instead of the actual toruqe vector.

Completely analogous to the default stiffness, the message

```
(send tg :define-trq 'nom-trq trq-vect)
```

will cause the VAX to use `trq-vect` as the default nominal torque for segs without torque specifications.

### 7.6.2 Finger Stiffness and Force

The current stiffnesses at the finger tips can be set by sending the message

```
(send tg :set-finger-stf stf-mat)
```

where `stf-mat` is a 9×9 *finger stiffness matrix.*

This message implements the following force/displacement relation

$$\begin{bmatrix} \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_3 \end{bmatrix} = K \begin{bmatrix} \Delta\underline{x}_1 \\ \Delta\underline{x}_2 \\ \Delta\underline{x}_3 \end{bmatrix}$$

where $\underline{f}_i$ and $\Delta\underline{x}_i$ are respectively the force and displacement vectors of the $i^{th}$ finger tip. The interpretation of the frame in which to implement the stiffness relation follows the rules for finger motions. For a basic-frame trajectory generator, the directions of the forces and displacements are interpreted as specified in the hand-fixed frame. For a grasp-frame trajectory generator, the directions are interpreted as specified in the body-fixed frame. The `back-to-basic` message can be sent if we wish to specify the stiffness relation in a hand-fixed frame. In the special case where the stiffness relation is to be specified in the hand frame, the optional :in-hand-frame argument can be used, i.e.

```
(send tg :set-finger-stf stf-mat :in-hand-frame T) .
```

Specifying only the stiffness will cause the default nominal torque to be applied. Nominal finger forces can be specified by the inclusion of another key-worded argument, or by a sending a separate `set-finger-frc` message. For example,

```
(send tg :set-finger-stf stf-mat :nom-frc frc-vect) .
```

or

```
(send tg :set-finger-frc frc-vect)
```

will set the nominal finger forces at the current knot point to `frc-vect`. Specifying only the finger forces will cause the default joint stffness to be used.

Analogous to the joint space counterparts, the finger stiffness and/or force at a previous knot point is set by including the optional `:seg-number` argument, e.g.

```
(send tg :set-finger-stf stf-mat :seg-number 8) .
```

The stiffness will be interpreted as specified in the body-fixed frame corresponding to that particular knot point, i.e. the stiffness is specified in the instantaneous object coordinates. This feature provides a very natural way to specify the stiffnesses and forces of fingers. For example, assuming that we are constructing a new trajectory, the following two different sequences of messages produce the the same effect:

Sequence 1:

```
(send tg :move-fingers-to grasp-position :in-hand-frame T)
(send tg :set-finger-stf stf-mat)
(send tg :generate-traj move1 :number-of-segs 9)
(send tg :set-finger-stf stf-mat)
```

Sequence 2:

```
(send tg :move-fingers-to grasp-position :in-hand-frame T)
(send tg :generate-traj move1 :number-of-segs 9)
```

```
(send tg :set-finger-stf stf-mat :seg-number 0)
(send tg :set-finger-stf stf-mat)
```

Again, this feature can be by-passed using the back-to-basic message.

### 7.6.3 Object Stiffness and Force

The generalized grasp stiffness at the current knot point is specfied by sending the sequence of two messages

```
(send tg :update-grasp-matrix)
(send tg :set-grasp-stf stf-mat) .
```

This message implements the generalized grasp force/dispacement relation

$$\begin{bmatrix} \underline{f} \\ \underline{m} \\ \underline{g} \end{bmatrix} = K \begin{bmatrix} \Delta\underline{x} \\ \Delta\underline{\theta} \\ \Delta\underline{d} \end{bmatrix}$$

where $\underline{f}$ and $\underline{m}$ are the net force and moment vector on the object and $\underline{g}$ is the internal grasp force vector. The vector $\Delta\underline{d}$ corresponds to the change in the distance between the grasp points.

The interpretation of desired compliance frame is completely analogous to that for setting the finger stiffness. For a basic-frame trajectory generator, the stiffness relation is interpreted as specified in the hand-fixed frame. For a grasp-frame trajectory generator, the stiffness is interpreted as specified in the body-fixed frame, i.e. the body-fixed frame is the compliance frame. The back-to-basic message can be sent if we wish to specify the stiffness relation in a hand-fixed frame. The optional :in-hand-frame argument can be used for the special case of setting the hand frame as the compliance frame.

It is interesting to note that a update-grasp-matrix message is sent prior to setting the stiffness, i.e. the grasp matrix is not updated automatically as part of the set-grasp-stf procedure. Recall that a grasp matrix is only a function of the positions of the contacts *relative* to the compliance frame. Hence, if the compliance frame is a body-fixed frame, then the grasp matrix need only to be

computed once for each grasp. That is, the grasp matrix need not be updated for every knot point at which we wish to specify the stiffness. Therefore, to avoid unnecessary computation, the task of updating the grasp frame is left to the programmer. The grasp matrix need to be re-computed only in two situations — when regrasping an object, and when the compliance frame is not a body-fixed frame.

The optional arguments to the `set-grasp-stf` message are also completely analogous to those for the `set-finger-stf` message. Nominal grasp force can be specified by

```
(send tg :set-grasp-stf stf-mat :nom-frc frc-vect)
```

or by sending a separate message

```
(send tg :set-grasp-frc frc-vect) .
```

The stiffness and/or force at a previous knot point is set by the optional `:seg-number` argument, e.g.

```
(send tg :set-grasp-stf stf-mat :nom-frc frc-vect :seg-number 8) .
```

Again, the stiffness relation will be interpreted as specified in the body-fixed frame corresponding to that knot point. This feature allows the programmer to naturally specify the desired body-centered stiffness at any point in the trajectory.

## 7.7 Sending a Trajectory

As messages are received by the trajectory generator, a joint trajectory with the associated stiffnesses and nominal torques is constructed. At any time, the programmer can either continue to construct the trajectory or execute the current trajectory. The trajectory can be executed by sending the message

```
(send tg :send-traj pc) .
```

where pc is the parallel connection object.

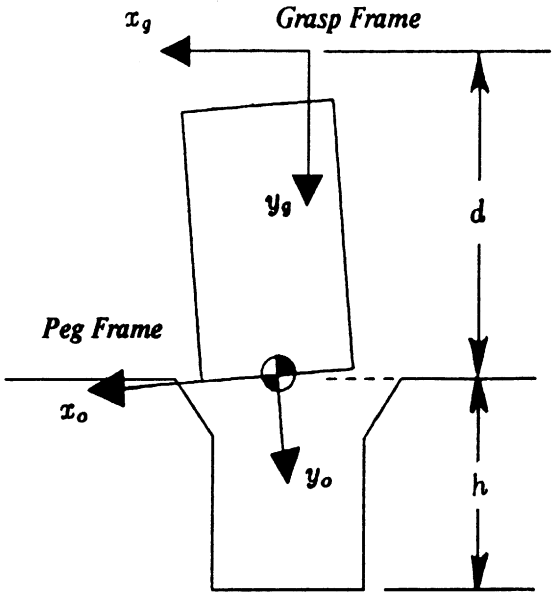



Figure 7.4: Peg-in-hole insertion using body-fixed compliance frame

When this message is received, the trajectory generator constructs message packets with the appropriate op-codes and arguments. As each packet is constructed, the trajectory generator sends a message to the parallel connection object, requesting that the packet be sent to the VAX. When all packets have been sent, the internal trajectory structure is cleared for storing a new trajectory.

## 7.8 Programming Examples

As an illustration of how these messages are used, we will consider a peg-in-hole insertion task using two different compliance frame specifications. The first will use a body-fixed compliance frame, while the second will use a hand-fixed compliance frame.

### 7.8.1 Peg Insertion Using Body-fixed Compliance

Consider the task of inserting a peg into a chamfered hole in the presence of alignment errors (see Figure 7.4). This task can be accomplished by allowing the geometric constraints imposed by the chamfer and the hole to guide the

motion of the peg. We will first consider using a body-fixed compliance during the insertion, as that achieved by the RCC device.

To place the compliance center at the tip of the peg, we define a body-fixed frame with the tip as origin, as shown in Figure 7.4. Assume that the axis of the peg is approximately aligned with the axis of the grasp frame, and that the distance from the origin of the grasp frame to the tip of the peg is approximately $d$, the peg frame relative to the grasp frame is given by

```
(setq peg-frame (make-frame NIL (vector 0.0 d 0.0))) .
```

A message is sent to the grasp-frame trajectory generator to set the body-fixed frame.

```
(send tg :set-frame peg-frame)
```

Let the initial vertical position of the peg tip be the same as the mouth of the hole, the task can then be achieved by the following sequence of messages:

```
(setq insert-y (list (list NIL NIL (vector 0.0 h 0.0))))
(send tg :generate-traj insert-y :number-of-segs 10)
(send tg :update-grasp-matrix)

(dotimes (i 10)
   (send tg :set-grasp-stf stf-mat :nom-frc frc-vect :seg-number i))

(send tg :send-traj pc)
```

where the `dotimes` structure is a simple do-loop, repeating the enclosed statements 10 times, with the variable `i` incremented from 0 to 9. The stiffness matrix `stf-mat` is usually a diagonal matrix, and the nominal force vector `frc-vect` usually specifies only internal grasp forces to be non-zero.

### 7.8.2 Peg Insertion Using Hand-fixed Compliance

Kinematic and force analysis [Whitney 1982] has shown that the ideal location of the compliance center for peg-in-hole insertions is at the mouth of the hole. However, because the RCC is a passive device, the compliance center has to

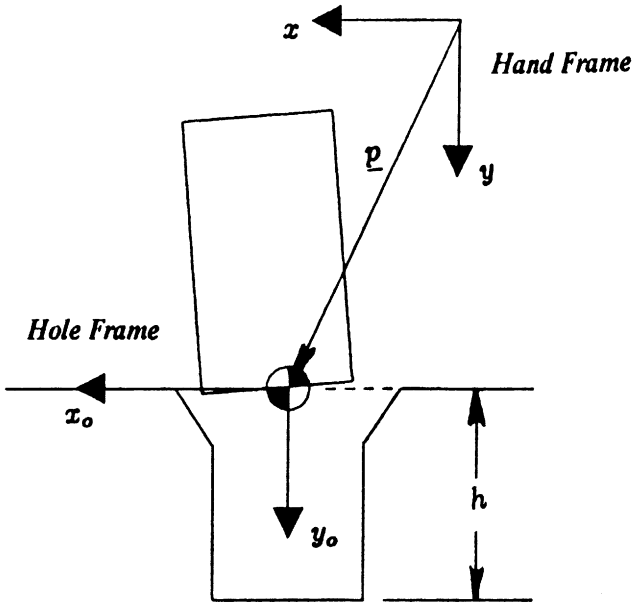



Figure 7.5: Peg-in-hole insertion using hand-fixed compliance frame

remain fixed with respect to the peg, and hence moves beyond the mouth of the hole during insertions. Using active stiffness control, we can specify the compliance center to remain fixed with respect to the hole.

The compliance frame defined at the hole is shown in Figure 7.5, having the same orientation as the hand frame. The vector $\underline{p}$ denotes the translation of the origin of the compliance frame from the origin of the hand frame. Therefore, the compliance frame relative to the hand frame is given by

```
(setq hole-frame (make-frame NIL p-vector)) .
```

To specify compliance in a hand-fixed frame, we first convert the grasp-frame trajectory generator into a basic-frame type.

```
(send tg :back-to-basic)
(send tg :set-frame hole-frame)
```

The task can then be achieved by the following sequence of messages:

```
(setq insert-y (list (list NIL NIL (vector 0.0 h 0.0))))
(send tg :generate-traj insert-y :number-of-segs 10)
```

```
(dotimes (i 10)
   (send tg :update-grasp-matrix)
   (send tg :set-grasp-stf stf-mat :nom-frc frc-vect :seg-number i))

(send tg :send-traj pc)
```

Note that the grasp matrix is now re-computed at every knot point.

The trajectory generator may well be implemented with a settable internal variable named auto-grasp-matrix-update. However, because the evaluation of a grasp matrix is computationally expensive, the syntax is designed to encourage using a body-fixed compliance frame when possible.

*This empty page was substituted for a blank page in the original document.*

# Chapter 8

# Conclusions

## 8.1 Review

This work was primarily concerned with coordinating the motion of an articulated hand to perform useful manipulations of objects. We began with an study of the nature of manipulation. It was shown that motions of rigid objects are determined by kinematic constraints. Manipulation corresponds to using the contacts to impose appropriate constraints such that the possible object motions will be uniquely the desired motion. The question of whether consistent kinematic constraints can be imposed led to the discussion of force control.

We then studied the kinematic transformations of articualted hands which translate the desired Cartesian motions of contacts into motions in joint space. The transformations for the Stanford/JPL Hand were derived as an example. These transformations were determined on the assumptions that the fingers do not slip and that rolling at the contacts is negligible. The geometry of the contacts were used to define a grasp frame, which provided the necessary link between the position and orientation of a grasped object and the joint coordinates of the hand. It was shown that the goal position and orientation of an object must be specified in terms of an object trajectory. This is a fundamental distinction between articulated hands and ordinary manipulators. The necessity to specify a trajectory path led to a discussion of how motions are specified,

and how points on a rigid object are located at each instant of the motion. Combining motion specification and the transformations developed earlier, we were able to translate object motions in arbitrary coordinate frames into a set of corresponding joint motions. However, joint space interpolations inevitably result in kinematic inconsistencies. In practice the inconsistencies may be absorbed by the mechanical compliance of the fingers. Still, this can result in excessive contact forces causing permanent deformations of the fingers and the object. Hence, regardless of whether the environment imposes any kinematic constraints, force control is necessary to safely resolve the internal kinematic inconsistencies between the fingers and the object.

The stiffness control strategy was considered. An analysis of the stiffness matrix showed how a stiffness matrix can be specified to obtain the desired compliant behavior. In essence, symmetric stiffness matrices can be used to orient the principal stiffness axes, and non-symmetric matrices can be used to obtain cross-coupled compliance. To implement stiffness control with an articulated hand, we must be able to translate the desired grasp forces into required joint forces. The transpose of the Jacobian provides the transformation from contact force to joint force, and hence a transformation from grasp force to contact force is required. This transformation is defined as the grasp matrix transpose. The product of the grasp matrix and the Jacobian matrix was defined as the grasp Jacobian. Assuming small deviations, the grasp Jacobian can be used to transform a Cartesian stiffness into a pre-computable joint stiffness. The results can be extended to include full impedance control, transforming Cartesian inertia, damping, and stiffness matrices into the joint space equivalents.

Using the stiffnes approach, compliant motion of objects then involves superimposing stiffness control on a desired nominal trajectory. This is implemented by setting the appropriate joint stiffnesses and bias forces at the trajectory knot points. Although trajectories and forces can be interpolated linearly or quadratically between knot points in joint space, it is not certain that the same holds for stiffness matrices. If the knot points are closely spaced, these simple interpolation methods can be used to eliminate undesirable oscillations or sudden

jumps in motion during stiffness transitions.

The choice of compliance frame basically affects the interpretation of the translational displacement of an object and the rotational force on the object. It should be chosen to reflect the "natural" definitions of translational displacement and rotational force for an object. In most cases, a natural choice is the body-fixed frame which defines the position and orientation of an object. Using a body-fixed frame as the compliance frame is also desirable from a computational point of view.

The compliant motion concepts and methodologies culminated in the implementation of a high-level hand programming sytem for the Stanford/JPL Hand. The hand programming system is basically an extension to an existing LISP system. LISP procedures are defined for computations specific to the hand and for communicating with a VAX dedicated to real-time control. High-level hand programming is realized through the implementation of two abstract objects, a trajectory generator, and a parallel connection object. Compliant motions are specified by sending messages to the trajectory generator, which constructs a joint trajectory with the associated stiffnesses and bias torques. Motions and stiffnesses can be specified at the joint, finger, or object level. At any time, the programmer can execute the current trajectory or continue to build the trajectory. The interactive nature of the system simplifies programming and debugging. The use of LISP as the basis of this system also supports integration of knowledge-based programs with manipulation.

## 8.2 The Future

The current development of the hand programming system has not reached the stage where conditionals can be sent as part of a trajectory. To check if a force threshold has been exceeded, the LISP machine must send a message to the VAX requesting the current force state. The goal of the immediate future is to implement the conditionals as part of a trajectory, such that the VAX will automatically respond with a message if a certain condition has occurred.

The conditionals will have different levels of priority. Certain conditions will completely pause the trajectory until further instruction is received, while others may simply send a message to the LISP machine.

Force sensors have been developed which can locate point contacts as well as measure normal and tangential forces at the contacts [Salisbury 1984b]. These sensors have recently been miniaturized as hemi-spherical finger tips to be used on the Stanford/JPL Hand. In the present hand programming system, joint motions are computed on the assumption that contacts occur at the tip of the fingers, neglecting rolling of the finger tips across the surface of the object. More accurate modeling of the grasp kinematics can be obtained from the sensor information. Pending the completion of the sensor system, algorithms can be developed to improve the dexterity of the hand. Forces exerted on an object is currently computed from the joint torques. The sensors will also provide more precise measurement of these forces. The use of tactile sensor information in a force feedback algorithm is an interesting topic of future research.

The current vocabulary of the hand control language is sufficiently broad to implement complex regrasping or exploration algorithms. Performing regrasping with a three-fingered hand is difficult, since two fingers with point contacts are insufficient to completely constrain an object. When a finger is withdrawn for regrasping, the object is free to rotate about the line connecting the two remaining fingers. The two finger grasp configuration must counteract the moments exerted by the gravity force. This can be achieved by using soft finger tips which are capable of exerting moments at the contacts as well as forces. The extra stability is obtained at the expense of positioning accuracy. An alternative is to push the object against the palm while a finger is re-positioned. However, it is not clear that free rotation of the object is undesirable. The gravity force may be exploited to manipulate the object into a stable configuration. A method for using local tactile sensor information and appropriate choice of finger stiffness to perform stable regrasp of two-dimensional objects was presented by Fearing [1984]. Regrasping in three-dimensions remains a new area of research. Works on interpreting tactile information for identifying objects has

also been limited. An algorithm for identifying polyhedral objects from local measurements of positions and surface normals was presented by Grimson and Lozano-Pérez [1984]. This requires an integration of manipulation, sensing, and model-based reasoning. The hand programming system is a step toward these goals.

Significant progresses have been made in recent years in the development of manipulator control, sensor technology, vision systems, and knowledge-based programs. Advances in these areas are made by researchers specialized in the particular fields. It is important to keep a broad perspective on the available technology. As the human hand compliments the eye, it may not be necessary to develop each technology to perfection for a particular application, but to integrate them properly to compliment each other.

*This empty page was substituted for a blank page in the original document.*

# Appendix A

# CADR/VAX Messages

This appendix lists the messages available for CADR/VAX communication. Message packets are constructed from two op-codes followed by appropriate data.

## 1. OP-CODE DEFINITIONS

### Op-codes for CADR to VAX Communication

| Symbol | Octal Number | Message |
|---|---|---|
| PAUSE-SYNC | #010 | pause servo control program |
| RESUME-SYNC | #020 | resume servo control program |
| START-TRAJECTORY | #030 | start trajectory execution |
| CHANGE-MODE | #040 | change control mode |
| CHAR-CMD | #060 | special character command |
| QUIT | #070 | quit |
| DEFINE-POS | #0100 | define position by a symbol |
| DEFINE-TRQ | #0200 | define torque by a symbol |
| DEFINE-STF | #0300 | define stiffness by a symbol |
| ADD-SEG | #01000 | add pre-defined position to trajectory |
| $ADD-SEG | #01100 | add position to trajectory |
| SET-TRQ | #02000 | set pre-defined torque |
| $SET-TRQ | #02100 | set torque |
| SET-STF | #03000 | set pre-defined stiffness |
| $SET-STF | #03100 | set stiffness |

*This empty page was substituted for a blank page in the original document.*

| | | |
|---|---|---|
| SET-COND | #04000 | set conditional, not implemented |
| QUERY-POS | #010000 | query position |
| QUERY-TRQ | #020000 | query torque |
| QUERY-FINGER-FRC | #030000 | query finger force |

### Op-codes for VAX to CADR Communication

| Symbol | Octal Number | Message |
|---|---|---|
| CHAR-MSG | #01 | character string message |
| INT-MSG | #02 | integer message |
| FLOAT-MSG | #03 | float message |
| CURRENT-POS | #010 | current position |
| CURRENT-TRQ | #020 | current torque |
| CURRENT-FINGER-FRC | #021 | current finger force |
| CURRENT-STATUS | #030 | current status |

## 2. MESSAGE PACKET SYNTAXES

### CADR to VAX Packet Syntaxes

The symbol OP-2 indicates that the second op-code is
not yet defined for the message; any integer is acceptable.

```
(PAUSE-SYNC OP-2)
(RESUME-SYNC OP-2)
(START-TRAJECTORY OP-2)
(CHANGE-MODE OP-2 CHAR)
(CHAR-CMD OP-2 NCHAR STRING)
(QUIT OP-2)
(DEFINE-POS OP-2 NCHAR NAME-STRING 12-FLOATS)
(DEFINE-TRQ OP-2 NCHAR NAME-STRING 12-FLOATS)
(DEFINE-STF OP-2 NCHAR NAME-STRING 90-FLOATS)
(ADD-SEG OP-2 NCHAR NAME-STRING DURATION)
($ADD-SEG OP-2 12-FLOATS DURATION)
(SET-TRQ OP-2 NCHAR NAME-STRING)
($SET-TRQ OP-2 12-FLOATS)
(SET-STF OP-2 NCHAR NAME-STRING)
```

*This empty page was substituted for a blank page in the original document.*

```
($SET-STF OP-2 90-FLOATS)
(QUERY-POS QUERY-ID)
(QUERY-TRQ QUERY-ID)
(QUERY-FINGER-FRC QUERY-ID)
```

## VAX to CADR Packet Syntaxes

```
(CHAR-MSG OP-2 STRING)
(INT-MSG OP-2 INTEGERS)
(FLOAT-MSG OP-2 FLOATS)
(CURRENT-POS REPLY-ID 12-FLOATS)
(CURRENT-TRQ REPLY-ID 12-FLOATS)
(CURRENT-FINGER-FRC REPLY-ID 9-FLOATS)
(CURRENT-STATUS REPLY-ID 12-INTEGERS)
```

*This empty page was substituted for a blank page in the original document.*

# Appendix B

# The Trajectory Generator

This appendix lists the messages which can be sent to a trajectory generator object. The first symbol is the operation name, followed by the arguments. The symbol &optional indicates that the remaining arguments are optional. The symbol &key indicates that the remaining arguments are optional arguments which must be preceeded by the appropriate keywords.

## 1. MESSAGES HANDLED BY ALL TRAJECTORY GENERATORS

```
(:INIT)
(:INIT-POS &OPTIONAL INITIAL-JOINT-POS CHG-DEFAULT-INT-POS)
(:SET-DEFAULT-SEGS-PER-MOVE SEGMENTS-PER-MOVE)
(:SET-DEFAULT-MOVE-DURATION DURATION-IN-SECONDS)
(:SET-DEFAULT-INTERNAL-POS POS-VECTOR)
(:SET-DEFAULT-INTERNAL-TRQ TORQUE-VECTOR)
(:SET-DEFAULT-INTERNAL-STF STIFFNESS-VECTOR)
(:SET-TRAJ-FILL-POINTER SEG-NUMBER)
(:SET-CURL CURL-OUT-VECTOR)
(:SET-FRAME FRAME)
(:SET-TRQ TORQUE-VECTOR-OR-SYMBOL &KEY SEG-NUMBER NAME)
(:SET-FINGER-FRC FORCE-VECTOR &KEY INTERNAL-TRQ SEG-NUMBER IN-HAND-FRAME NAME)
(:SET-GRASP-FRC FORCE-VECTOR &KEY INTERNAL-TRQ SEG-NUMBER IN-HAND-FRAME NAME)
(:SET-JOINT-STF STIFFNESS-MATRIX-OR-SYMBOL &KEY NOM-TRQ SEG-NUMBER NAME)
(:SET-FINGER-STF STIFFNESS-MATRIX &KEY NOM-FRC INTERNAL-STF
                                      SEG-NUMBER IN-HAND-FRAME NAME)
```

*This empty page was substituted for a blank page in the original document.*

```
(:SET-GRASP-STF STIFFNESS-MATRIX &KEY NOM-FRC INTERNAL-STF
                                      SEG-NUMBER IN-HAND-FRAME NAME)
(:UPDATE-GRASP-MATRIX &KEY SEG-NUMBER IN-HAND-FRAME)
(:DEFINE-POS SYMBOL-NAME &OPTIONAL SEG-NUMBER-OR-JOINT-POS)
(:DEFINE-TRQ SYMBOL-NAME TORQUE-VECTOR)
(:DEFINE-STF SYMBOL-NAME STIFFNESS-MATRIX)
(:NULL-MOVE &OPTIONAL DURATION)
(:MOVE-JOINTS-TO JOINT-POS-OR-SYMBOL-NAME &KEY DURATION)
(:MOVE-JOINTS-BY JOINT-POS &KEY DURATION)
(:MOVE-FINGERS-TO CARTESIAN-POS &KEY INTERNAL-POS DURATION IN-HAND-FRAME)
(:MOVE-FINGERS-BY CARTESIAN-POS &KEY INTERNAL-POS DURATION IN-HAND-FRAME)
(:GENERATE-TRAJ MOVE-SPEC-LIST &KEY NUMBER-OF-SEGS DURATION IN-HAND-FRAME)
(:FIND-RELATIVE-POS)
(:COPY-TRAJ &OPTIONAL FLOAT-BUF)
(:SEND-TRAJ PARALLEL-CONNECTION)
```

## 2. ADDITIONAL MESSAGES HANDLED BY GRASP-FRAME TYPE

```
(:ORIENT-WITH ORIENTATION-FRAME &KEY NUMBER-OF-SEGS DURATION)
(:MOVE-TO GOAL-FRAME &KEY NUMBER-OF-SEGS DURATION)
(:BACK-TO-BASIC)
(:SET-AUTO-FRAME-UPDATE T/NIL)
(:SET-ORIGIN-AT-CENTROID T/NIL)
(:UPDATE-GRASP-FRAME)
```

*This empty page was substituted for a blank page in the original document.*

# Appendix C

# The Parallel Connection

This appendix lists the messages which can be sent to a parallel connection object. The parallel connection object was implemented by Patrick A. O'Donnell.

### MESSAGES HANDLED BY ALL PARALLEL CONNECTIONS

```
(:INIT)
(:KILL)
(:DATA-AVAILABLE)
(:GET-NEXT-PKT &OPTIONAL WAIT?)
(:MAY-TRANSMIT)
(:SEND-PKT PACKET)
```

*This empty page was substituted for a blank page in the original document.*

# References


1. **ASEA**, "Industrial Robot System," ASEA AB, Sweden, YB 110-301 E.
2. **Brooks, T.L.**, "Optimal Path Generation for Cooperating or Redundant Manipulators," *Proc. 2nd Int. Computer Engineering Conference*, San Diego, CA, August 1982, pp 119-122.
3. **Drake, S.**, " Using Compliance in Lieu of Sensory Feedback for Automatic Assembly," Charles Stark Draper Laboratory Report T-657, Sept. 1977.
4. **Fearing, R.S.**, "Simplified Grasping and Manipulation with Dextrous Robot Hands," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, AI Memo 809, Nov. 1984.
5. **Goldman, R.**, "Design of an Interactive Manipulator Programming Environment," Ph.D. Thesis, Department of Computer Science, Stanford University, Dec. 1982.
6. **Grimson, W.E. and T. Lozano-Pérez**, "Model-based Recognition and Localization from Tactile Data," *Proc. IEEE Int. Conference on Robotics*, Atlanta, GA, March, 1984.
7. **Hogan, N.**, "Impedance Control of Industrial Robots," Robotics and Computer-Integrated Manufacturing, Vol 1, No. 1, 1984, pp 97-113.
8. **Hollerbach, J.M. and K.C. Suh**, "Redundancy Resolution of Manipulators through Torque Optimization," *Proc. IEEE Int. Conference on Robotics and Automation*, St. Louis, MO, March 1985.
9. **Holt, H.R.**, "Robot Decision Making," Cincinnati Milacron Inc., MS77-751, 1977.

10. **Kazerooni, H.**, "A Robust Design Method for Impedance Control of Constrained Dynamic Systems," Ph.D. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Feb. 1985.

11. **Mason, M.T.**, "Compliance and Force Control for Computer Controlled Manipulators," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, AI TR 515, April 1979.

12. **Paul, R.P.**, "Manipulator Path Control," *Proc. IEEE Int. Conference on Cybernetics and Society*, New York, Sept. 1975, pp 147-152.

13. **Paul, R.P.**, "WAVE: A Model-based Language for Manipulator Control," *The Industrial Robot*, March 1977.

14. **Paul, R.P.**, "Manipulator Cartesian Path Control," *IEEE Trans. on Systems, Man, Cybernetics*, SMC-9, 1979, pp 702-711.

15. **Paul, R.P.**, *Robot Manipulators*, M.I.T. Press, Cambridge, MA, 1981.

16. **Paul, R.P. and Shimano B.**, "Compliance and Control," *Proc. Joint Automatic Control Conference*, Purdue University, July, 1976.

17. **Raibert, M.H., and J.J. Craig**, "Hybrid Position/Force Control of Manipulators," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol 102, June 1981, pp. 126-133.

18. **Roberts, L.G.**, "Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs," Lincoln Laboratory, Massachusetts Institute of Technology, Document No. MS1045, 1965.

19. **Salisbury, J.K.**, "Active Stiffness Control of a Manipulator in Cartesian Coordinates," *Proc. 19th IEEE Conference on Decision and Control*, Albuquerque, NM, Dec. 1980.

20. **Salisbury, J. K.**, "Kinematic and Force Analysis of Articulated Hands," Ph.D. Thesis, Department of Mechanical Engineering, Stanford University, May 1982.

21. **Salisbury, J.K.**, "Design and Control of an Articulated Hand," *Proc. 1st Int. Symposium on Design and Synthesis*, Tokyo, July 1984a.

22. **Salisbury, J. Kenneth**, "Interpretation of Contact Geometries from Force Measurements", *Proc. 1st International Symposium on Robotics Research*, Bretton Woods, NH, MIT Press, Sept. 1984b.

23. **Salisbury, J.K. and J.D. Abramowitz**, "Design and Control of a Redundant Mechanism for Small Motion," *Proc. IEEE Int. Conference on Robotics and Automation*, St. Louis, MO, March 1985.

24. **Salisbury, J.K. and J.J. Craig**, "Articulated Hands: Force Control and Kinematic Issues," *Int. Journal of Robotics Research*, Vol 1, No. 1, Spring 1982.

25. **Silver, D.**, "The Little Robot System," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, AI Memo 273, Jan. 1973.

26. **Talylor, R.H.**, "Planning and Execution of Straight-line Manipulator Trajectories," *IBM Journal of Research and Development* 23, 1979, pp 424-436.

27. **Talylor, R.H., P.D. Summers, and J.M. Meyer**, "AML: A Manufacturing Language," *Robotics Research* 1, 3, Fall, 1982.

28. **Whitney, D. E.**, "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators," *ASME Journal of Dynamic Systems, Measurement, and Control*, Dec. 1972, pp. 303-309.

29. **Whitney, D. E.**, "Force Feedback Control of Manipulator Fine Motions," *ASME Journal of Dynamic Systems, Measurement, and Control*, June 1977, pp. 91-97.

30. **Whitney, D. E.**, "Quasi-Static Assembly of Compliantly Supported Rigid Parts," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 104, March 1982, pp. 65-77.

31. **Winston, P.H. and B.K. Horn**, *LISP*, Addison-Wesley Publishing Co., Inc., 1981.

*This blank page was inserted to preserve pagination.*

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>AI-TR 1029 | 2. GOVT ACCESSION NO.<br>AD-A196893 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>Generating Compliant Motion of Objects with an Articulated Hand | | 5. TYPE OF REPORT & PERIOD COVERED<br>technical report<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Stephen L. Chiu | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00014-82-K-0494 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Artificial Inteligence Laboratory<br>545 Technology Square<br>Cambridge, MA 02139 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Advanced Research Projects Agency<br>1400 Wilson Blvd.<br>Arlington, VA 22209 | | 12. REPORT DATE<br>June 1985<br>13. NUMBER OF PAGES<br>134 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)<br>Office of Naval Research<br>Information Systems<br>Arlington, VA 22217 | | 15. SECURITY CLASS. (of this report)<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

articulated hands
coordinated manipulation
stiffness control
robot programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The flexibility of the robot is the key to its success as a viable aid to production. Flexibility of a robot can be expanded in two directions. The first is to increase the physical generality of the robot such that it can be easily reconfigured to handle a wide variety of tasks. The second direction is to increase the ability of the robot to interact with its environment, such that tasks can still be successfully completed in the presence of uncertainties. The use of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

articulated hands offers a means for expanding the flexibility of the robot in both directions. Articulated hands are capable of adapting to a wide variety of grasp shapes, hence reducing the need for special tooling. The availability of low mass, high bandwidth joints close to the manipulated object also offers significant improvements in the control of fine motions. This thesis provides a framework for using articulated hands to perform local manipulation of objects. In particular, it addresses the issues in effecting compliant motions of objects in Cartesian space. The Stanford/JPL Hand is used as an example to illustrate a number of concepts. The examples provide an unified methodology for controlling articulated hands grasping with point contacts. We also present a high-level hand programming system based on the methodologies developed in this thesis. Compliant motion of grasped objects and dextrous manipulations can be easily described in the LISP-based hand programming language.

# Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

**Scanned**

**Date:** 09/18/1995

**M.I.T. Libraries**
**Document Services**