University of Pennsylvania
ScholarlyCommons

September 1986

Design and Implementation of a Robot Force and Motion Server

Hong Zhang
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Hong Zhang, "Design and Implementation of a Robot Force and Motion Server", . September 1986.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-86-73.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/672
For more information, please contact repository@pobox.upenn.edu.

Design and Implementation of a Robot Force and Motion Server

Abstract

A robot manipulator is a force and motion server for a robot. The robot, interpreting sensor information in terms of a world model and a task plan, issues instructions to the manipulator to carry out tasks.

The control of a manipulator first involves motion trajectory generation needed when the manipulator is instructed to move to desired positions. The procedure of generating the trajectory must be flexible and efficient. When the manipulator comes into contact with the environment such as during assembly, it must be able to comply with the geometric constraints presented by the contact in order to perform tasks successfully. The control strategies for motion and compliance are executed in real time by the control computer, which must be powerful enough to carry out the necessary computations.

This thesis first presents an efficient method for manipulator motion planning. Two fundamental modes of motion, Cartesian and joint, are considered and transition between motion segments is uniformly treated to obtain an efficient and simple system. A modified hybrid control method for manipulator compliance is then proposed and implemented. The method overcomes the problems existing in previous approaches such as stiffness control and hybrid control. Finally, a controller architecture is studied to distribute computations into a number of processors to satisfy the computational requirement in a cost-effective manner. The implementation using Intel's single board computers is also discussed. Finally, to demonstrate the system, the motion trajectory. and the modified forced/motion control scheme are implemented on the controller and a PUMA 260 manipulator controlled from a multi-user VAX/Unix system through an Ethernet interface.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-86-73.

This technical report is available at ScholarlyCommons: https://repository.upenn.edu/cis_reports/672

DESIGN AND IMPLEMENTATION OF A ROBOT FORCE AND MOTION SERVER

Hong Zhang

MS-CIS-86-73
GRASP LAB 77

Department Of Computer and Information Science
Moore School
University of Pennsylvania
Philadelphia, PA 19104

September 1986

Acknowledgements: This research was supported in part by DARPA grants NOOO14-85-K-0018 and NOOO14-85-K-0807, NSF grants DMC-8411879, DMC-85-12838, DCR-86-07156, DCR8501482, MCS8219196-CER, MCS-82-07294, 1 RO1-HL-29985-01, U.S. Army grants DAA6-29-84-K-0061, DAAB07-84-K-F077, U.S. Air Force grant 82-NM-299, AI Center grants NSF-MCS-83-05221, U.S. Army Research office grant ARO-DAA29-84-9-0027, Lord Corporation, RCA and Digital Equipment Corporation.

DESIGN AND IMPLEMENTATION
OF
A ROBOT FORCE AND MOTION SERVER

by

Hong Zhang

A Thesis

Submitted to the Faculty

of

Purdue University

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 1986

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my appreciations to those who have helped me throughout my graduate study to make this work possible. Professor Richard P. Paul, the Chairman of my graduate committee, has initiated the ideas on which the research described in this writting is based. His direction, encouragement and inspiration are essential to the success of the research and, at the same time, have made these years such a pleasant experience.

My special thanks go to National Science Foundation for its financial support, to the Computer Integrated and Design Manufacture Automation Center of Purdue University for providing the facilities in the Robotics Laboratory and the financial support, and to General Robotics and Active Sensory Perception Laboratory and the Department of Computer and Information Science, University of Pennsylvania, for providing access to the facilities in conducting the research and preparing of this manuscript.

I thank my committee members, Professors Shaheen Ahmad, Owen Mitchell, and Daniel Gottlieb, in spending their time on suggesting research alternatives and reading the thesis.

Finally, I would like to thank all professors whom I have been educated by and to the students and colleagues whom I have been associated with, directly and indirectly, at the School of Electrical Engineering and other Schools of Purdue University, for the times of sharing, learning, and making a dream come true.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF SYMBOLS ix

ABSTRACT xii

CHAPTER I - INTRODUCTION 1

1. Introduction 1
2. Manipulator Motion Control 2
3. Force Control by Manipulators 4
4. Controller Architecture 6
5. Research Objectives 7
6. Organization 7

CHAPTER II - BACKGROUND 9

1. Introduction 9
2. Homogeneous Transformations 9
3. Kinematics of Robot Manipulators 11
4. Differential Relationships 14
5. Static Force and Torque Transformation 16

CHAPTER III - MOTION CONTROL OF ROBOT MANIPULATORS 18

1. Introduction 18
2. Previous Approaches 19
3. A Unified Motion Control Strategy 22
    3.1. Coordinate Systems 23
    3.2. Description of Position 24
    3.3. Description of Motion 25

Page

3.4. Trajectory Generation 26
3.4.1. General Consideration 27
3.4.2. Joint Coordinate Motion 29
3.4.3. Cartesian Coordinate Motion 32
3.5. Discussion 36
4. Conclusion 42

CHAPTER IV - CALCULATION OF MANIPULATOR DYNAMICS 43

1. Introduction 43
2. Formulation of Dynamics Equations 44
3. Comparisons of the Two Formulations 47
4. Simplification of Lagrangian Dynamics Equations 48
4.1. Gravity Loadings 49
4.2. Effective Inertias 49
4.3. Coupling Inertias 50
5. Determination of the Dynamics Constants 50
5.1. Joint Torque Calibration 51
5.2. Determination of Gravity Loadings 53
5.3. Effective Inertias 56
6. Conclusion 60

CHAPTER V - COMPLIANCE AND HYBRID CONTROL 61

1. Introduction 61
2. Compliance Specification 63
3. Representatives of Active Force Control 65
4. Modified Hybrid Control Method 71
5. Implementation 75
6. Stability Consideration 79
6.1. State Space Formulation 80
6.2. Special Case 83
6.3. Discussion 87
7. Conclusion 87

CHAPTER VI - INTEGRATION OF THE RFMS 89

1. Introduction 89
2. General Consideration 92

Page

3. Identification of the RFMS Processes 94
    3.1. User Process 96
    3.2. Communication Process 99
    3.3. Kinematic Process 100
    3.4. Dynamic Process 101
4. Implementation of the RFMS 102
    4.1. User Level 106
    4.2. Ethernet Communication 106
    4.3. Supervisor 107
    4.4. Joint Processors 111
    4.5. Math Process 113
    4.6. Direct Memory Access 114
    4.7 Performance Evaluation 114
5. Conclusion 115

CHAPTER VII - CONCLUSION AND FUTURE WORK 118

1. Summary 118
2. Future Work 119

LIST OF REFERENCES 121

APPENDICES

Appendix A: Kinematics of PUMA 250 Manipulator 127
Appendix B: Dynamics Equations of PUMA 560 Manipulator 141
Appendix C: Distributed Trajectory Generator and Hybrid Controller 144

VITA 152

LIST OF TABLES

Table Page

3.1. Boundary Conditions for h 30
3.2. Boundary Conditions for θ 31
3.3. Motion Control Summary 37
4.1. Joint Torque Calibration of PUMA 560 53
4.2. Configurations for c_{ij} Measurement 55
6.1. Multibus Address Assignment 104
6.2. Supervisor-Joint Interaction 111

Appendix
Table

A1. PUMA 260 Link and Joint Parameters 127
A2. Differential Transformations 133
A3. Radii of Gyration for the PUMA 560 141

LIST OF FIGURES

Figure Page

1.1. Peg-in-hole Operation 2

2.1. Interpretation of a Homogeneous Transformation 10

3.1. Segments of Motion 27

3.2. Transition Polynomials 29

3.3. Trajectories of Joint One in Joint Mode 39

3.4. Joint Mode Trajectories with Longer t_{acc} 40

3.5. Trajectories of Joint One in Cartesian Mode 41

4.1. Joint Torque versus Motor Current 52

4.2. Joint Torque Calibration for the PUMA 560 54

5.1. Tracing a Corner 65

5.2. A Pure Position Controller 66

5.3. Stiffness Control Method 67

5.4. Free-joint Method 68

5.5. Hybrid Control Method 70

5.6. A Simple Manipulator 71

5.7. Modified Hybrid Control Method 74

5.8. MHCM Controller Design 76

6.1. Levels of Control in the RFMS 95

6.2. The RFMS Implementation 103

6.3. Development System 105

Page

6.4. Memory Organization 108
6.5. Position Ring Structure 110
6.6. Process Scheduling 116

LIST OF SYMBOLS

Symbol	Description
$\mathbf{A}_i$	Matrices relating the position and orientation of link i of the manipulator to link $i+1$.
$\mathbf{a}$	approach vector of a homogeneous transformation
a_i	length of link i.
$\mathbf{C}_\theta$	joint compliance matrix = $\mathbf{J}^{-1}\bar{S}\mathbf{J}$
$\mathbf{D}$	drive transformation used in deriving straight-line motion of a manipulator.
D_i	gravity loading of joint i in the Lagrangian dynamics equation of a manipulator.
D_{ij}	coupling inertia between joints i and j when $i \neq j$; effective inertia of joint i when $i=j$.
D_{ijk}	centripetal and Coriolis coefficients.
d_i	distance between two neighboring joints.
$\mathbf{F}$	force vector representing Cartesian forces of the manipulator with three scalar force components and three torque components.
$\mathbf{f}$	desired Cartesian biased force vector of six elements representing three forces along and three torques about x, y, and z axes, respectively.
f_s	sample frequency in Hertz.
h	motion parameter and a function of time.

I	actuator inertia of a joint.
$\mathbf{J}$	Jacobian matrix of the manipulator, which is $6\times n$ where n is the number of joints.
$\mathbf{J}^{-1}$	inverse Jacobian matrix.
$\mathbf{K}$	diagonal Cartesian stiffness matrix.
$\mathbf{K}_\theta$	diagonal joint stiffness or positional gain matrix.
$\hat{\mathbf{K}}_\theta$	joint stiffness matrix in Salisbury's stiffness control.
$\mathbf{M}$	mode structure describing how a motion is to be achieved.
m	point mass of the load being carried by the manipulator.
$\mathbf{n}$	normal vector of a homogeneous transformation
$\mathbf{o}$	orientation vector of a homogeneous transformation
P, P_k	labels identifying position equations.
$\mathbf{p}$	position vector of a homogeneous transformation
r	$1-h$
$\mathbf{S}$	a six-by-six Cartesian compliance selection matrix.
$\mathbf{T}_6$	a four-by-four matrix representing the position of the manipulator end relative to its base.
t	time variable.
$\mathbf{v}$	Cartesian velocity $= [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$ obtained by taking derivatives of $\mathbf{x}$ with respect to time.
$\mathbf{x}$	Cartesian position of the manipulator when represented as a vector three translational and three rotational variables.
α_i	twist angle between two neighboring links.

Δt	sampling period of the joint servos.
$\Gamma()$	procedure to solve a position equation for T_6.
$\Lambda()$	kinematics of the manipulator.
$\boldsymbol{\omega}$	Cartesian rotational velocity vector = $[\omega_x, \omega_y, \omega_z]^T$.
$\boldsymbol{\tau}$	joint torque vector representing torques being exerted by the joints.
$\boldsymbol{\theta}$	joint position vector = $[\theta_1, \theta_2, \cdots, \theta_n]^T$.
$\dot{\boldsymbol{\theta}}$	joint velocity = $\frac{d\boldsymbol{\theta}}{dt}$.
$\ddot{\boldsymbol{\theta}}$	joint acceleration = $\frac{d\dot{\boldsymbol{\theta}}}{dt}$.
$\dddot{\boldsymbol{\theta}}$	joint jerk = $\frac{d\ddot{\boldsymbol{\theta}}}{dt}$.
$d\boldsymbol{\theta}$	joint position error vector = $[d\theta_1, d\theta_2, \cdots, d\theta_n]^T$.
$d\boldsymbol{\theta}_{ri}$	real position error used in the hybrid control.

ABSTRACT

Zhang, Hong. Ph.D., Purdue University, May 1986. Design And Implementation Of A Robot Force And Motion Server. Major Professor: Richard P. Paul.

A robot manipulator is a force and motion server for a robot. The robot, interpretating sensor information in terms of a world model and a task plan, issues instructions to the manipulator to carry out tasks.

The control of a manipulator first involves motion trajectory generation needed when the manipulator is instructed to move to desired positions. The procedure of generating the trajectory must be flexible and efficient. When the manipulator comes into contact with the environment such as during assembly, it must be able to comply with the geometric constraints presented by the contact in order to perform tasks successfully. The control strategies for motion and compliance are executed in real time by the control computer, which must be powerful enough to carry out the necessary computations.

This thesis first presents an efficient method for manipulator motion planning. Two fundamental modes of motion, Cartesian and joint, are considered and transition between motion segments is uniformly treated to obtain an efficient and simple system. A modified hybrid control method for manipulator compliance is then proposed and implemented. The method overcomes the problems existing in previous approaches such as stiffness control and hybrid control. Finally, a controller architecture is studied to distribute computations into a number of processors to satisfy the computational requirement in a cost-effective manner. The implementation using Intel's single board computers is also discussed. Finally, to demonstrate the system, the motion trajectory and the modified force/motion control scheme are implemented on the controller and a PUMA 260 manipulator controlled from a multi-user VAX/Unix system through an Ethernet interface.

CHAPTER I

INTRODUCTION

1. Introduction

The study of robotics deals with the intelligent interaction between machines and the physical world. Through such interactions, a robot performs useful tasks, collecting information from the world, understanding it, and altering the world as required by the task. A manipulator acts as a force and motion server (RFMS) to a robot in task executions. It moves with or without objects along defined paths, manipulates objects in the world, and applies forces or torques to objects.

It is fundamental that a manipulator is able to move to an arbitrary position within the workspace and change between positions along desired paths, and that it is able to apply forces and torques in arbitrary directions during the motion. It is equally important that during a motion a manipulator be able to react to changes observed by the sensors such as contacts forces between the manipulator and the environment or positions of objects being manipulated. These requirements are necessary for a robot manipulator to assist a robot system to interact with the world effectively.

Consider, for example, a peg-in-hole operation (Figure 1.1) in an assembly task, the robot calls for the vision system to locate the peg and the hole and then asks the manipulator to pick up the peg, move it to an appropriate approach position, make contact with hole with an approach angle with a force sensor monitoring if the contact is made, and, once making the contact, tilt the peg toward the principle axis of the hole while maintaining zero force and zero torques in the plane perpendicular to the hole axis. If, however, the hole is moved accidentally, the vision system tracking positions immediately informs the robot system and the approach and final positions are properly adjusted before the manipulator is instructed to proceed.

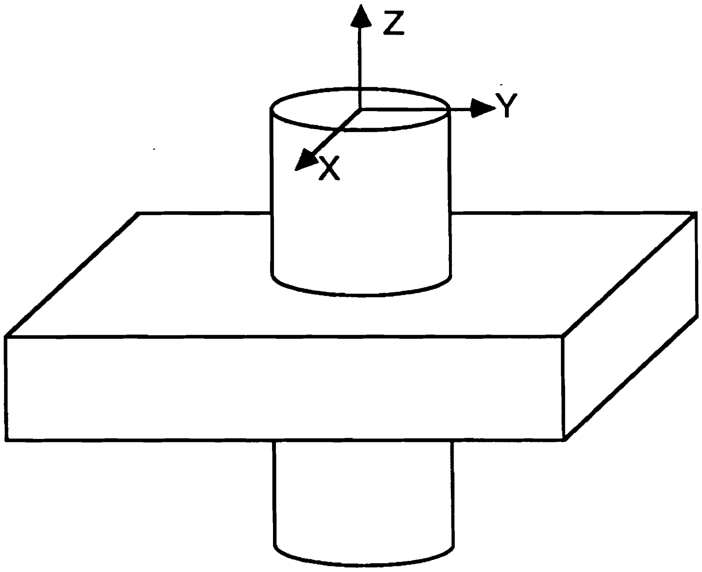


Figure 1.1. Peg-in-hole Operation

2. Manipulator Motion Control

The earlist manipulators were controlled in a master-slave manner with the operator positioning the master and a slave imitating the action of the operator [Goertz 1952]. This mechanism was widely applied to handling of dangerous materials in hostile environment to protect people from direct contact with hazards; unfortunately, the manipulator requires the full-time attendance of an operator. The idea supporting such manipulators, nevertheless, is still used today in remotely controlled robot manipulators in space or underwater exploration.

The use of computers in manipulator controllers enabled teach-and-repeat operation in which the manipulator remembers what it is taught by recording the successive joint positions and then repeats the trajectory thereafter. This kind of operations can be performed by the manipulators accurately, because errors can be reduced to minimum during the calibration procedure and robot manipulators have a good repeatability. In the case of mass production such as on an assembly line, these robot manipulators have proved to be extremely useful due to their accuracy, reliability and durability. However, these robot manipulators are inadequate in situations where tasks are of the small-batch nature, or when the manipulators are required to be sensitive to external changes while performing tasks.

In recent years, the increased requirement on the sophistication of robot manipulator performance and rapid growth of computer technology gave rise to the construction of many programming systems for robot manipulator control. These control systems, although different in implementation, provide important features necessary for programming robot manipulators to undertake complicated tasks. Manipulator level robot programming systems have been extensively studied and applied to robot manipulator control [Shimano, Geschke, and Spalding 1984] [Hayward and Paul 1984]. Task level systems are receiving more and more attentions and many systems exist [Lieberman and Wesley 1976]. In general, such a control system provides programming languages in which tasks can be mathematically expressed, provides sensor interfaces to modify a motion while the motion is being executed, and provides tools for program development and simulation [Karel].

There are a number of issues which must be addressed by every manipulator control system. It must be able to specify all motions achievable by the manipulator in a consistent and concise fashion. It must be able to generate specified motion trajectories in terms of the coordinates in which a manipulator is controlled. Methods used must be computationally efficient for real-time execution.

3. Force Control by Manipulators

For a robot manipulator to be useful, it must be able to control not only positions and but also forces. It must be able to comply to geometric constraints while performing a task. Such an ability is important when forces need be applied by the manipulator or when geometric constraints presented by a task make it impossible for the manipulator to succeed with its limited positional accuracy. A majority of the industrial robots working in factories today perform repetitive tasks such as spray painting or spot welding what require relatively low positional accuracy. When contact is inevitable such as in assembly operation, they employ passive mechanical compliance devices [Whitney 1982] or special jigs and textures to overcome the inability. However, such a method is not applicable nor cost-effective if a robot needs to perform many tasks of different constraints.

An alternative approach for a manipulator to be able to comply to geometric constraints is to program it to react to motion constraints when they occur. The manipulator complies in directions where geometric constraints are observed and remains rigid in the other directions. Since local relative measurements are used and, more importantly, since the manipulator uses force, which is usually of larger magnitude and therefore easier to detect than the positional errors, control performance can be improved significantly. Difficulty arises, though, when a manipulator is actuated at joints and the compliance behavior is required in the Cartesian space. A solution must be found to convert Cartesian compliance specification to joint actuation adjustments.

Adequate mathematical formulation and successful force control methods, such as active stiffness control [Salisbury 1980] and hybrid control [Raibert and Craig

1981], have been developed in the past few years. Based on many previous works, Mason formally summerized the problem of compliance [Mason 1979]. He makes use of ideal compliance surface and introduces selection matrix to partition the degrees of freedom available into two orthogonal complementary sets. One set corresponds to directions of natural constraints, those presented by the task; the other corresponds to directions of artificial constraints, those imposed by the control system to achieve a desired trajectory.

Based on these theoretical developments, the stiffness control was proposed by Salisbury [1982] who argued that a manipulator can be programmed to act as a six dimensional spring with respect to the compliance frame, having high stiffnesses in unconstrained directions and low stiffnesses in constrained directions. Cartesian reaction forces are computed based on the stiffnesses and Cartesian positional errors. Joint reaction forces then are easily obtained by converting the Cartesian reaction forces. This approach allows variable stiffness specification as a task proceeds, and proves to be satisfactory in many applications.

Hybrid control method proposed by Craig and Raibert [1981] considers a robot manipulator as a positioner, except that position errors in constrained directions should be tolerated or ignored. The selection matrix specifies just what errors should be ignored. One can then compute Real Cartesian position errors, from which real joint position errors are easily obtained for the joint servos. This method is computational more expensive than the stiffness method since the inverse Jacobian matrix is needed, but eliminates some of the problems that exist in the stiffness method.

Most of the implementations of force controllers were of experimental nature, controlling a subset of the three dimensional space or ignoring some important considerations such as computation efficiency of the control algorithms and their impact on the stability of the resulting systems.

4. Controller Architecture

The computations required to control a robot manipulator are performed by computers in the manipulator controller. The advances of microprocessor technology have made it possible to design cost-effective yet powerful computing systems to meet the needs of controllers employing sophisticated control algorithms.

Parallel computer architecture is widely used in designs of robot manipulator controllers. Almost all the industrial robots are equipped with multi-processor controllers to distribute computations to a number of processors [Unimation 1980]. These control systems employ a low Cartesian set-point loop and a high joint servo rate to perform inverse kinematics and interpolate the solutions. Such a design can generate smooth motions, which are desired for pick-and-place operations in industry, but it leads to long time delay from a modified Cartesian set-point to adjusted joint actuations. Fast sensor-driven motions is unstable in such systems and the control system designs usually do not fully consider the issue of sensor integration. Controllers have also been designed in research laboratories to control robot manipulators for specific applications. These attempts are of experimental nature without considering the cost-effectiveness.

The progresses in micro-processor technology have made it possible now to design robot manipulator controllers of powerful processing capability at a moderate cost. More research attentions are also being paid to integration of a completion robot system equipped with a number of sensors, as opposed to an independent robot manipulator [Giralt 1984, Taylor 1985] [Paul and Durrant-Whyte 1986]. The role of a manipulator in the robot system and its control are yet to be studied.

5. Research Objective

This research studies motion and force control strategies for a robot manipulator. It examines basic formalisms for manipulator motion control and force control and proposes and implements new methods. The method for obtaining the simplified model of the manipulator dynamics is also introduced. All control techniques developed emphasize the computational efficiency for real-time control applications. Finally, the problem of system integration is investigated. A manipulator control system capable of providing force and motion services to a robot system is constructed based on a distributed computer architecture. The system executes commands received from the robot coordinator. The system is designed to facilitate sensor-driven motions.

6. Organization

Chapter II summerizes background mathematics necessary for developing theories in later chapters. The useful properties of homogeneous transformation are studied and homogeneous transformation is then used as a basic tool for representation of relationships between objects and definitions of positions in the robot environment. The Chapter provides important relationships and formulas that are used throughout the later chapters.

Chapter III describes the trajectory generation or motion planning. It starts with the classification of motions and their specifications. An efficient method of trajectory generation is then introduced, to provide two fundamental types of motion, Cartesian and joint, and uniformly treats transitions between two segments of motion.

Chapter IV examines the dynamics of a robot manipulator. The relative significance of dynamics coefficients in Lagrangian formulation of dynamics equations is studied. Based on the conclusion that gravity loadings, effective and some coupling inertias represent a dynamics model with sufficient accuracy, a method is introduced to experimentally determine the constants in those terms in the simplified dynamics equations. The method can be easily applied to all electric actuated manipulators.

Compliance and hybrid control is discussed in Chapter V. Different force control methods are reviewed and compared. Some existing problems are also pointed out. Based on the discussions, a modified hybrid control method is presented. The method is efficient computationally and is implemented to control a PUMA 560 manipulator. Stability analysis is performed to show the problem associated with this method and many other Cartesian control techniques when the remote center of compliance is required. Solution to the problem is also explored.

Chapter VI integrates the manipulator control system and studies its construction as a robot force and motion server (RFMS), using a distributed computer architecture. The processes for the RFMS are specified and then classified into dynamic, kinematic, and static modules. An assignment of processes to processors can proceed based on the real-time computation requirements of the processes. Finally the Chapter describes the implementation of this server using Intel single board computers.

In the last chapter, major conclusions of the the research described in the thesis are outlined and future work based on the research is suggested.

CHAPTER II

BACKGROUND

1. Introduction

It is important to highlight some aspects of mathematical basis for the study of robot manipulation before the presentation of the thesis. Although various theories of mathematics have been applied for different purposes, a system based on homogeneous transformations has proved to be the most appropriate in terms of its completeness and efficiency [Paul 1983]. All the discussions in this chapter will make use of such a system.

A coordinate frame can be associated with an object in the robot work space. Control of a robot consists of determining the geometric relationships between objects and changing the relationships in the way defined by tasks. There are six degrees of freedom in a three dimensional space, three translational and three rotational. These relationships must then be descriptions of the six degrees of freedom.

2. Homogeneous Transformations

A relationship between two coordinate frames can be represented by a *homogeneous transformation* of the form:

$$\mathbf{T} = [\mathbf{n}, \mathbf{o}, \mathbf{a}, \mathbf{p}] = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

home components of **p** correspond to changes in translation in x, y, and z directions. The **n, o, a** directional unit vectors in three directions of the new coordinate frame relative to the first frame. This is illustrated in Figure 2.1.

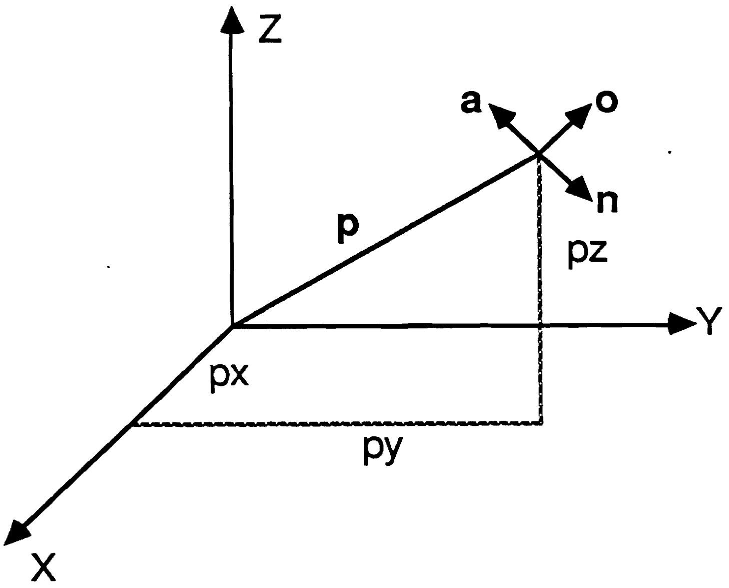


Figure 2.1. Interpretation of a Homogeneous Transformation

Homogeneous transformations provide the property instrumental in coordinate frame composition: the composition of two coordinate relationships corresponds to the matrix product of the two homogeneous transformations. If $\mathbf{T}_1$ describes coordinate frame **B** with respect to frame **A** and $\mathbf{T}_2$ describes frame **C** with respect to frame **B**, then **T** describes frame **C** with respect to frame **A** where

$$\mathbf{T} = \mathbf{T}_1 * \mathbf{T}_2 \tag{2.2}$$

In addition, the composition of two homogeneous transformations is also a homogeneous transformation. The inverse of a homogeneous transformation describes a relationship in the reverse direction to the forward transformation and it can be calculated easily by

$$\mathbf{T}^{-1} = \begin{bmatrix} n_x & n_y & n_z & -\mathbf{p}\cdot\mathbf{n} \\ o_x & o_y & o_z & -\mathbf{p}\cdot\mathbf{o} \\ a_x & a_y & a_z & -\mathbf{p}\cdot\mathbf{a} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

where "." stands for vector dot product.

3. Kinematics of Robot Manipulators

A manipulator is a mechanical linkage consisting of a set of links connected by one of the lower pairs. The most common lower pairs used in manipulators today are prismatic or translational and revolute or rotary, each one of which provides one degree of freedom translating along or rotating about a joint.

If a manipulator is to be positioned and oriented arbitrarily, it must possess at least six joints to match the degrees of freedom in three dimensional space. The description of the position and orientation of the last link, $\mathbf{T}_n$ or the Cartesian position of the manipulator, can be obtained symbolically from the parameters and variables defining the n links and n joints [Hartenburg and Denavit 1964]. The procedure to obtain $\mathbf{T}_n$ consists of assigning a coordinate frame to each link, formulating matrices, conventionally called $\mathbf{A}_i$ for each ith link, which define the relationship between ith and $(i\text{-}1)$th link, and performing coordinate frame composition by multiplying the A matrices.

For a revolute manipulator, each ith link is defined by the link length a_i and the link twist angle α_i, and the ith joint is defined by the joint distance d_i and the joint angle θ_i [Paul 1981]. A prismatic joint can be defined similarly with joint angle θ_i being a constant and joint distant d_i being variable.

Once the parameters and variables are obtained, $\mathbf{A}_i$ matrix is defined as

$$\mathbf{A}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The Cartesian position and orientation of a manipulator is then

$$\mathbf{T}_n = \mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_n \tag{2.5}$$

The process of determining the Cartesian position of a manipulator from joint positions is referred to as the *direct kinematics* and the joint positions uniquely determine $\mathbf{T}_n$.

Each element of $\mathbf{T}_n$ is, in general, a function of all joint variables.

$$\mathbf{T}_n = \begin{bmatrix} f_{11}(\boldsymbol{\theta}) & f_{12}(\boldsymbol{\theta}) & f_{13}(\boldsymbol{\theta}) & f_{14}(\boldsymbol{\theta}) \\ f_{21}(\boldsymbol{\theta}) & f_{22}(\boldsymbol{\theta}) & f_{23}(\boldsymbol{\theta}) & f_{24}(\boldsymbol{\theta}) \\ f_{31}(\boldsymbol{\theta}) & f_{32}(\boldsymbol{\theta}) & f_{33}(\boldsymbol{\theta}) & f_{34}(\boldsymbol{\theta}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_n]$. However, most of the manipulators have six joints arranged as a three joint positioning mechanism followed by an orienting mechanism or a wrist of another three joints in order to simplify the design and control. Consequently, the position vector $\mathbf{p} = [f_{14}, f_{24}, f_{34}]^T$ are functions of only the first three positioning joints, i.e.,

$$f_{i4}=f_{i4}(\theta_1, \theta_2, \theta_3) \tag{2.7}$$

The *inverse kinematic solution* is defined as the process of determining joint positions from a given $\mathbf{T}_n$. This solution is important as a manipulator task, specified in Cartesian positions, must be transformed into joint coordinates where the control is exercised. The inverse solution, different from the direct solution, is not unique in general, i.e., a given Cartesian position can be satisfied with more than one set of joint positions in general, each set corresponding to one manipulator *configuration.* Furthermore, not all the manipulators have closed-form solutions to explicitly express joint variables in terms of the elements of $\mathbf{T}_n$. When this happens, the inverse kinematics must be solved numerically using the 12 equations

$$\begin{aligned} f_{11} &= n_x, f_{21} = n_y, f_{31} = n_z \\ f_{12} &= o_x, f_{22} = o_y, f_{32} = o_z \\ f_{13} &= a_x, f_{23} = a_y, f_{33} = a_z \\ f_{14} &= p_x, f_{24} = p_y, f_{34} = p_z \end{aligned} \tag{2.8}$$

This is a non-linear overdetermined system, which can be solved by various numeric techniques[Angeles 1985]. The solution, however, is usually much more computationally expensive than a closed-form solution [Pieper 1968].

Systematic procedures exist for solving inverse kinematics for manipulators for which there exist closed-form solutions [Paul, R.P. and Zhang, H. 1986]. For a six joint manipulator, the procedure solves the joints recursively from joint one to joint six using the following equations

$$\begin{aligned} \mathbf{T}_6 &= \mathbf{V}_0 = \mathbf{U}_1 \\ \mathbf{A}_1^{-1}\,\mathbf{T}_6 &= \mathbf{V}_1 = \mathbf{U}_2 \\ \mathbf{A}_2^{-1}\,\mathbf{V}_1 &= \mathbf{V}_2 = \mathbf{U}_3 \\ \mathbf{A}_3^{-1}\,\mathbf{V}_1 &= \mathbf{V}_3 = \mathbf{U}_4 \\ \mathbf{A}_4^{-1}\,\mathbf{V}_3 &= \mathbf{V}_4 = \mathbf{U}_5 \\ \mathbf{A}_5^{-1}\,\mathbf{V}_4 &= \mathbf{V}_5 = \mathbf{U}_6 \end{aligned} \tag{2.9}$$

where $\mathbf{U}_i = \mathbf{A}_i\mathbf{A}_{i+1} \cdots \mathbf{A}_6$. In these equations, quantities on the left-hand side are known and ith equation solves for joint i. Such a solution is usually very efficient compared to the numeric solutions.

Given in Appendix A are the modeling of a PUMA 250 manipulator in terms of its link and joint parameters, its direct kinematic solution, and its inverse kinematic solution. The manipulator, typical of industrial robot manipulator, consists of six revolute joints, possesses three configurations, and has a closed-form inverse kinematics [Paul and Zhang 1986].

4. Differential Relationships

The time derivatives of positions and orientations of a coordinate system correspond to its linear and angular velocities in and about respectively axes. Given velocities or differential changes in one coordinate system, their equivalent in another coordinate system can be calculated if the transformation relating the two systems is known. Assume velocities are expressed in a vector with three linear components and three angular components

$$\mathbf{v} = [v_x,\ v_y,\ v_z,\ \omega_x,\ \omega_y,\ \omega_z]^T \tag{2.10}$$

then the velocities in the coordinator frame which relates to the first by **T** can be found by:

$$\begin{aligned}
{}^{T}v_x &= \mathbf{n}\cdot((\boldsymbol{\omega}\times\mathbf{p})+\mathbf{v})\\
{}^{T}v_y &= \mathbf{o}\cdot((\boldsymbol{\omega}\times\mathbf{p})+\mathbf{v})\\
{}^{T}v_z &= \mathbf{a}\cdot((\boldsymbol{\omega}\times\mathbf{p})+\mathbf{v})\\
{}^{T}\omega_x &= \mathbf{n}\cdot\boldsymbol{\omega}\\
{}^{T}\omega_y &= \mathbf{o}\cdot\boldsymbol{\omega}\\
{}^{T}\omega_z &= \mathbf{a}\cdot\boldsymbol{\omega}
\end{aligned} \tag{2.11}$$

where the **n**, **o**, **a**, and **p** vectors are the respective columns of the transformation relating the two frames, "×" stands for vector cross product, and the leading superscript ^{T} stands for the frame being referenced [Paul 1981].

The above equation can be used to obtain the relationship between joint velocities and the velocities seen at the last link of the manipulator. This relationship, commonly known as *Jacobian matrix* **J**, can be calculated by determining the relationship of the velocities at the last coordinate frame to each of n joints, one at a time. Since the transformation from any joint to the end coordinate frame can be known from the A matrices and a joint has a one dimensional velocity, either angular or linear, the resultant velocities at the end coordinate frame due this joint velocity can be directly computed using Eq. (2.11). Once the individual results are obtained, the Jacobian matrix is simply the combination with its ith column for ith joint and has the general form

$$\mathbf{J} = [\frac{dx}{d\theta}, \frac{dy}{d\theta}, \frac{dz}{d\theta}, \frac{\delta x}{d\theta}, \frac{\delta y}{d\theta}, \frac{\delta z}{d\theta}]^T \tag{2.12}$$

The Cartesian velocities of a manipulator can then be obtained with

$$\mathbf{v} = \mathbf{J}\dot{\boldsymbol{\theta}} \tag{2.13}$$

where $\dot{\boldsymbol{\theta}} = [\dot{\theta}_1, \dot{\theta}_2, \cdots, \dot{\theta}_n]^T$. The inverse problem that, given the Cartesian velocities, find corresponding joint velocities can be solved obviously by

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}\mathbf{v} \tag{2.14}$$

The $\mathbf{J}^{-1}$ is the inverse of **J**, which is defined uniquely if **J** is square and $\det\mathbf{J} \neq 0$. The case of non-squared **J** is not considered here. When $\det\mathbf{J} = 0$, $\mathbf{J}^{-1}$ does not exist. This implies the manipulator loses one or more degrees of freedom at that position and is unable to move in certain Cartesian directions. Such configurations are defined as *singularity* or *degeneracy* points of the manipulator. In regions about the singularity points, excessive manipulator joint rates are required, making its control unstable.

A differential change vector

$$d\mathbf{x} = [d_x, d_y, d_z, \delta_x, \delta_y, \delta_z]^T \tag{2.15}$$

can be represented in the form of a transformation as

$$\Delta = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

When Δ is multiplied on the right by a transformation **T** representing a coordinate frame, the resulting transformation is the new differential change with respect to **T**, i.e.,

$$\Delta \mathbf{T} = \mathbf{T}^T \Delta \tag{2.17}$$

If there are a number of differential changes, the total differential change is equal to multiplication of their transformation representations,

$$\Delta = \Delta_1 \Delta_2 \cdots \Delta_n \tag{2.18}$$

5. Static Force and Torque Transformation

A generalized force vector can be applied to a coordinate frame with three forces along and three moments about the x, y, and z directions, respectively, corresponding to the six degrees of freedom in three dimensional space. The vector has the form

$$\mathbf{F} = [f_x, f_y, f_z, m_x, m_y, m_z]^T \tag{2.19}$$

The generalized force exerted at one coordinator frame can be transformed into an equivalent in another frame. Torques applied by the joints of the manipulator can also be transformed to a force vector at end of the manipulator or any other coordinate system in the manipulator. The transformation is derived using the concept of virtual work, work done by a force vector to cause differential displacements. The relationship is simple and expressed by the following equation.

$$\begin{aligned}\boldsymbol{\tau} &= \mathbf{J}^T\mathbf{F} \\ \mathbf{F} &= \mathbf{J}^{Tc}\mathbf{F}\end{aligned} \tag{2.20}$$

where $\tau = [\tau_1, \tau_2, \cdots, \tau_n]^T$ is the joint torque vector [Paul 1981].

CHAPTER III

MOTION CONTROL OF ROBOT MANIPULATORS

1. Introduction

A robot manipulator needs to be able to be arbitrarily positioned along desired paths and reach desired destinations within its work space. This Chapter studies the control of manipulator motions, their specification, generation, and execution. A method based on [Paul 1979] is proposed and implemented to uniformly deal with the various transitions between motion segments, resulting in an efficient and simpler system.

A robot manipulator motion is described as a sequence of positions through which the end of the manipulator is to pass. The description of the positions may be simple, as when manipulator joint coordinates are used, or complex, as, for example, when motion is referenced to some functionally defined coordinate frame. Motion between positions may be specified in detail or in general terms. When only the end points of the motion are of importance, the motion may be specified as joint coordinate or Cartesian coordinate motion. When the intermediate positions are of importance, then the motion may be specified by a procedure or by a table of coordinates.

A motion can be free of any actions when, for example, a manipulator moves parts or tools, or itself to a new position. A motion can also be accompanied by actions as a manipulator moves between positions. Spray painting and the application of sealants might be specified by a table of coordinates through which the applicator must pass. Seam welding along complex geometric paths, such as along the joint between two cylinders, might be functionally defined. Straight line seam welds could be defined by a coordinated Cartesian motion to which a sinusoidal weaving pattern is added. Motions to bring parts together in assembly operations might be defined simply by Cartesian coordinate motions. End effector actions take place while the manipulator is at rest. Finally, any of the above activities might be performed on a work piece which is in motion, such as on a conveyor.

Once a task is defined, the motion control system of a manipulator plans the motion so as to satisfy the motion specification. The evaluation of a motion planner is based on a number of criteria: adequate specification must exist to allow motions to be defined accurately and conveniently; the planning process must be efficient in computation and execution; concatenation between segments of motion must be provided to guarantee the continuity of the motion; and motions must be able to be modifiable by sensors during execution. The key considerations in designing a motion planning system have been generality and simplicity - it must be general enough to accommodate other control algorithms such as sensor-guided motion and compliant motion and must be efficient enough for real-time applications.

2. Previous Approaches

Various methods have been developed for obtaining trajectory generators. Whitney introduced *resolved motion rate control* [Whitney 1972], which makes use of inverse Jacobian matrix to continuously convert interpolated Cartesian positions to joint positions. Given an initial and a goal position, their Cartesian coordinates $\mathbf{x}$ in terms of three positions and three rotations can be derived. The choice of the coordinates for positions is obvious whereas the choice for rotations is not. When the initial and goal position are computed as $\mathbf{x}_i(\boldsymbol{\theta}_i)$ and $\mathbf{x}_f(\boldsymbol{\theta}_f)$, the difference

$$\Delta \mathbf{x} = \mathbf{x}_f - \mathbf{x}_i \tag{3.1}$$

can be computed. Given the segment time T of the motion, the Cartesian rate or velocity of the segment is computed as

$$\mathbf{v} = \frac{\Delta \mathbf{x}}{T} \tag{3.2}$$

The joint rates are computed using the differential relationship **J**, i.e.,

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}\, \mathbf{v}, \tag{3.2}$$

where **J** takes the tool frame into consideration. The joint positions are obtained by either reading the joint positions or integrating the velocity with respect to time as

$$\boldsymbol{\theta} = \boldsymbol{\theta}_i + \int_0^t \dot{\boldsymbol{\theta}}\, dt \tag{3.3}$$

To find proper rotation axis about which the Cartesian angular velocities are defined, he uses the eigenvector with the unit eigenvalue, ω, of the rotational transformation relating the initial and final Cartesian transformations, which remains unchanged during the rotation. The amount of rotation, α, is derived by projecting the x axes of the initial and final positions unto the plane perpendicular to ω. The angular velocities or the last three components of **x** can be defined as $\frac{\omega\, \alpha}{T}$. The above calculation is made closed loop by defining a new $\boldsymbol{\theta}_i$ periodically along the trajectory, hence a new T, ω and α.

Equation (3.2) requires a total of $6\times n$ multiplications and $5\times n$ additions for an n joint manipulator, when the inverse Jacobian is available. This approach has a number of drawbacks. The approach is not general enough to accommodate some other applications such as manipulator compliance. The transition between motion segments is not provided. The joint rate is computed using the current and final positions but the sampling period of the system is always a finite number. As the result, joint position deviates from the desired trajectory constantly.

Paul [1979] introduced a general approach to Cartesian trajectory generation based on homogeneous transformations. A *drive* transform, $\mathbf{D}(h)$, computed from the initial and goal position equations, is added to an equation describing the initial position in terms of the transformations in goal position and, as the motion variable h varies linearly from 0 to 1, the manipulator is brought from the initial position to the goal along a straight line and with two rotations. The successive Cartesian positions are converted to the joint coordinates through the inverse kinematics.

The **p** vector of the drive transform is the difference between the goal and the initial **p** vectors, i.e.,

$$\mathbf{p}_{\mathrm{D}} = \mathbf{p}_f - \mathbf{p}_i; \tag{3.4}$$

if this is interpolated by a scalar h linear with respect to time, a linear Cartesian motion is generated. The rotation part of the **D** transform represents a rotation of two linear components; the first is about a fixed axis and the second about the approach direction, **a**, of the tool transformation. The constants in $\mathbf{D}(h)$ are computed once only at the beginning of the motion and the intermediate positions need only to evaluate $\mathbf{D}(h)$ with the appropriate h. The intermediate positions require fewer numeric calculations than does updating an inverse Jacobian matrix used in resolved motion rate control. The frame in which motion occurs can be controlled easily by the location of $\mathbf{D}(h)$ in the position equation.

Transitions are considered in this approach separately for Cartesian and joint motions. There are four different kinds of transition in terms of the current and the next mode of motion since there are two possible modes. A transition to a Cartesian motion is performed in Cartesian space by a smoothing polynomial blending the current motion parameters in $\mathbf{D}(h)$ into the next segment. A transition to joint motion is performed separately by each joint by a joint space smoothing polynomial blending the two sets of boundary motion parameters in terms of joint positions, velocities, accelerations. Any transition involving two different modes is performed by first converting the current motion to a motion with the mode of the next motion so that the transition can be treated as the one of the previous two cases.

Taylor later refined Paul's method in several aspects [Taylor 1977]. Quaternion representation is used for the rotational transformation in order to reduce the amount of computation. For motions with constant initial and goal positions, intermediate positions can be precalculated sufficiently close to meet *bounded deviation* criteria. In real time, interpolation techniques can be then applied time using precalculated positions. The degeneracy problem is also treated to certain extent within the context of his approach.

3. A Unified Motion Control Strategy [Paul and Zhang 1985]

In Whitney's method large errors result when the manipulator moves at high speed, since the evaluation of the inverse Jacobian matrix requires too much computation. It is also difficult to use his method to generate sensor-driven motions. The transition in Paul's and complex and inefficient. In addition, the joint motion in his approach does not provide tracking of the final position as it should when the position is moving. A new approach is proposed and implemented in the following discussion. It maintains the generality in Paul's method, deals with the transition in an efficient manner, and provides tracking in the joint motion.

Transition occurs when a manipulator is about to complete the motion to the current destination position. Continuity of the motion is assured by the transition process in which the motion parameters change continuously to those of next motion segment. Transition can take place in joint space, in which case it is computed on a per joint basis, or in Cartesian space, in which case the Cartesian position of the manipulator is under control during the transition. The transition is very efficient in joint space while it is not in Cartesian space [Paul 1981]. However, if changes during the transition in joint space are kept small, changes in Cartesian space will also be small, confining the the Cartesian position to a bounded region, provided the manipulator is not close to a singularity. When the manipulator is close to the singularity, the motion as well as the transition is numerically unstable [Paul and Stevenson 1984b]. If transitions are handled exclusively in the joint space, however, the complexity of the transition is reduced. This observation thus serves as a basis for dealing with the transition in the following motion generation method. Further if joint motion is generated from the destination position back to the initial position, the tracking of the final position can be easily provided.

3.1. Coordinate Systems

There are two fundamental coordinate systems to describe manipulator positions, joint coordinates and Cartesian coordinates. The position of a manipulator is uniquely specified by the joint coordinates and it is in joint coordinates that a manipulator is controlled. Joint coordinates are, however, generally non-orthogonal with respect to Cartesian directions and, therefore, do not provide a convenient set of coordinates in which to perform the coordinate transformations used in the specification of manipulation. Cartesian coordinates, on the other hand, are convenient in this respect, but they must be mapped to joint coordinates to exercise control. The mapping process is done in real time and is computationally expensive. Converting Cartesian coordinates to joint coordinates, also known as inverse kinematics, is usually a one-to-many mapping, with one kinematically equivalent Cartesian configuration corresponding to a set of different joint configurations. In specifying a task, therefore, choices of desired configurations must be considered so that the inverse kinematics is reduced to a one-to-one mapping.

3.2. Description of Position

The end of an n joint robot manipulator is specified by the joint coordinates θ_n or by a homogeneous transformation $\mathbf{T}_n$ which specifies the Cartesian position and orientation of the end of the manipulator with respect to its base. A manipulator position may be specified by a *position equation*, which, in its simplest form, equates $\mathbf{T}_n$ to a homogeneous transformation of desired position:

$$\mathbf{T}_n = \mathbf{A}. \tag{3.5}$$

It is convenient to add more transformations to a position equation in order to express the structure of the position. For example,

$$\mathbf{T}_n\ \mathbf{Tool} = \mathbf{Obj}\ \mathbf{A} \tag{3.6}$$

indicates that a tool is attached to the end of the manipulator and the destination position is described relative to an object. This description makes the representation flexible, but the solution difficult as the equation must be manipulated to solve for $\mathbf{T}_n$ first before joint coordinates can be obtained.

The transformations in a position equation can be of different types. The transformation $\mathbf{T}_n$ is *read-only*. It is meaningless to assign values to it, for it is defined by the rest of the transformations in the position equation. A transformation can also be of type *value*. A *value* transformation is passed to the motion process with its present values, which will not change once a motion is commenced even if it may be assigned different values. The use of *value* transformations enables the motion control system to premultiply these transformations in a position equation, thus reducing the computational load during execution when the equation must be repeatedly evaluated.

Transformations whose values need be changed while motions involving these transformations are being executed correspond to another type, *ref* for reference. These transformations are used to provide a mechanism to incorporate sensory feedback, by which position and orientation of the manipulator is corrected or modified while it is in motion. A camera, for example, monitors the approach of the manipulator to position **A** in the above equation by correcting the values of **Obj**. The reference to a transformation can take place in the form of function evaluations, i.e., the components of the transformation are defined as functions of motion parameters so that, at different stages of the motion, the transformation renders different values.

3.3. Description of Motion

A manipulator task is defined by a sequence of position equations through which the manipulator must be moved. A label is associated with each equation and, when all the positions are solved, an ordered sequence of motions results and each of them is of the form:

$$P_i\colon \mathbf{T}_n = \mathbf{Expr}_i. \tag{3.7}$$

Manipulator tasks are specified by these positions in terms of Cartesian coordinates. Motions between these positions, however, can be either in joint coordinates or in Cartesian coordinates. After initial and goal positions are specified, other motion parameters determine characteristics of a motion. These parameters are related to time such as velocities and accelerations in Cartesian space or in joint space. Motion can proceed at a constant velocity or at a constant acceleration followed by constant deceleration. If motion parameters are specified as a structure of necessary attributes, **M**, a motion can then be initiated by a *move* function.

$$s = move(P_i, \mathbf{M}_i), \tag{3.8}$$

which means to move to position p_i with mode $\mathbf{M}_i$. This function returns a sequence number with a *move* request in order to simplify task synchronization.

3.4. Trajectory Generation

A *trajectory* of a manipulator is defined as a time sequence of manipulator positions in joint coordinates which brings the manipulator from its initial position to the goal position. These positions must be sufficiently close to each other, forming input to the manipulator servo system to control the manipulator smoothly and accurately. A *trajectory planner* or *trajectory generator* is defined as the process which generates these positions for a manipulator so as to satisfy such move requests as (3.7).

One fundamental requirement on a trajectory planner is the minimum rate at which the planner must supply *set-points* to the joint servos. Set-points include information on desired manipulator state in terms of joint positions, velocities, and accelerations. The joint coordinate motions involve moderate amount of computations for initial and goal joint positions, for interpolation of the intermediate joint positions, and for the inverse kinematics when tracking of the final position is required. Cartesian coordinate motions, on the other hand, are much more difficult to compute because of the complexity of inverse kinematics solved not only for the initial and goal positions, but also for all intermediate positions. Motion of either mode requires additional computations if the transformations in the destination position equation are not of *value* type, thus requiring evaluation of the goal position equation all the time. Efficiency of the employed technique for the trajectory planner is important for a system performance. An inefficient algorithm results in a decreased servo rate and a degraded system.

In addition to generating set-points for the intermediate positions, a trajectory planner must provide means for transition between segments of motion. Motion parameters such as mode and acceleration time for the current motion are usually not those for the next and, therefore, a blending trajectory must be generated to assure the continuity of position, velocity and acceleration of the trajectory.

3.4.1. General Consideration

The kinematics of a motion between two positions can be defined in terms of relative motion parameter h. The Cartesian position vector $\mathbf{x}$, which consists of three translational components and three rotational components, is computed by a function,

$$\mathbf{x} = \mathrm{f}(h). \tag{3.9}$$

The motion is mostly a constant velocity motion by virtue of h varying linearly with time, except for its beginning and end where transition takes place, as h varies in such a way that the motion dynamics are taken into account by

$$h = \mathrm{g}(t). \tag{3.10}$$

The constant velocity may apply either to joint space, as in joint motion, or to Cartesian space, as in Cartesian motion. Assume a general case illustrated in Figure 3.1, in which the manipulator is at position **A** at time $t = -t_{acc}$ and a motion is to be generated from the current destination position **B** to the next destination position C.

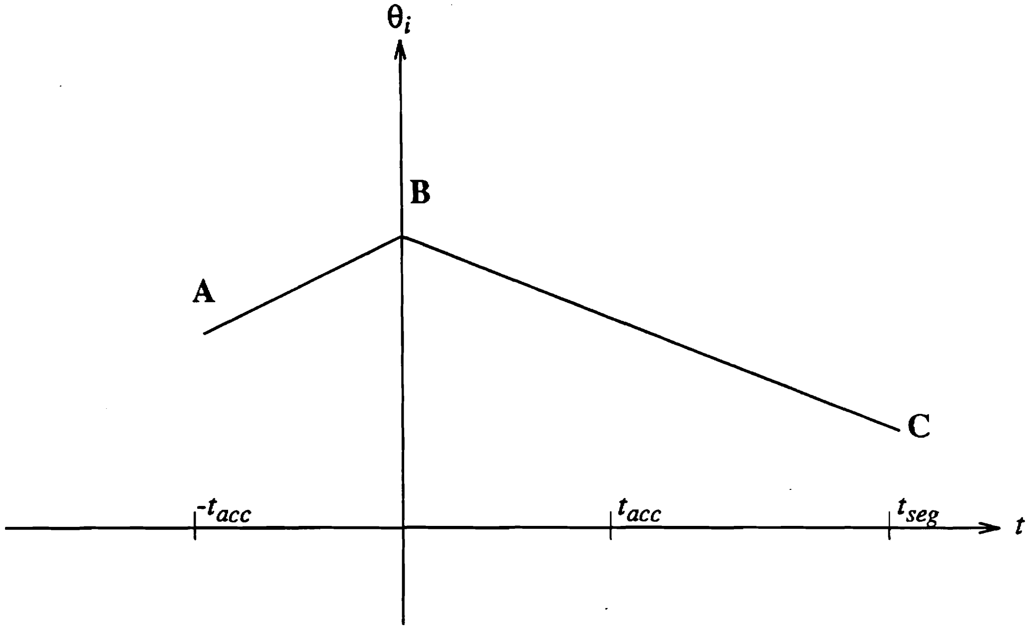


Figure 3.1. Segments of Motion

A segment of motion trajectory consists of two stages: in the first stage, the motion starts from rest at $-t_{acc}$ and accelerates until t_{acc}, at which time the trajectory has attained the desired velocity; in the second stage, the motion proceeds with the specified constant velocity until transition to the next segment of motion starts. This applies to both joint and Cartesian space. The motion trajectory consists of two parts: the first part, θ_{jnt_i} in case of joint motion and θ_{Car_i} in case of a Cartesian motion, specifies a motion starting from rest and finally moving at the specified velocity. If the manipulator is already in motion, a transition is required. At $-t_{acc}$, as the manipulator begins the execution of the first part, the second part, a *matching polynomial* θ_{mch_i}, is derived to remove any motion discontinuities between the current motion and next motion to **C**. The transition is performed in the joint space and the discontinuities are defined in the joint space regardless of the mode of motion in the next motion segment. The two parts of the trajectory between $-t_{acc}$ and t_{acc} are illustrated in Figure 3.2 for both Cartesian and joint motion.

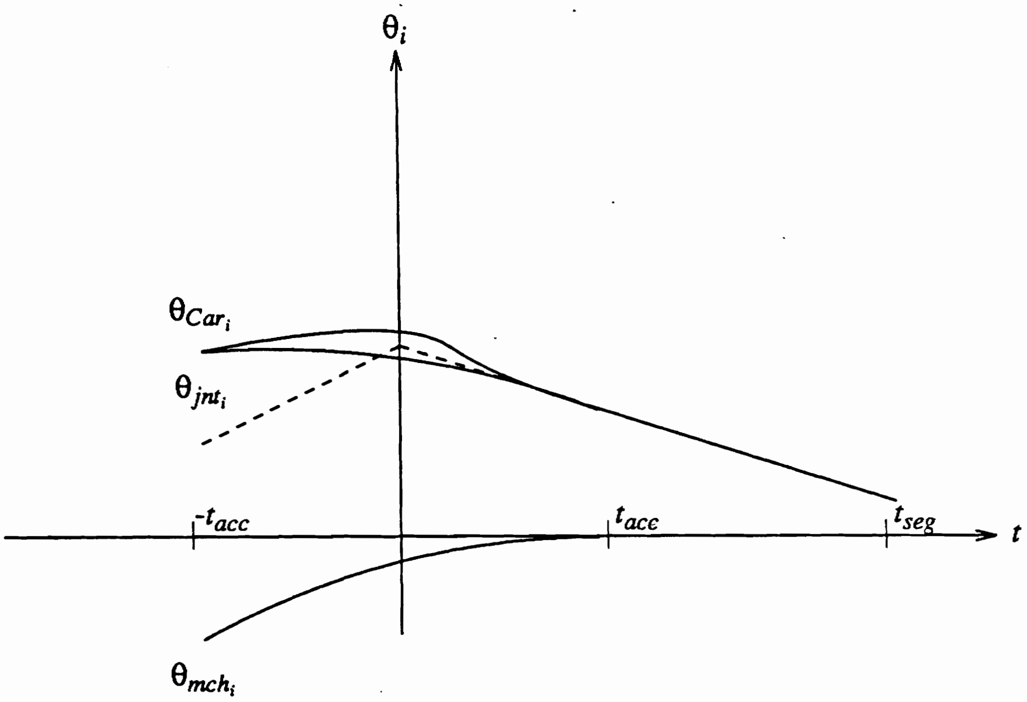


Figure 3.2. Transition Polynomials

To develop the mathematics for generating the above trajectories, let

$$\begin{aligned} \mathbf{B} &= \mathbf{Expr}_i(-t_{acc}) \\ \mathbf{C} &= \mathbf{Expr}_{i+1}(-t_{acc}), \end{aligned} \tag{3.11}$$

where the argument $(-t_{acc})$ indicates both positions are to be evaluated at the beginning the transition.

3.4.2. Joint Coordinate Motion

The motion from **B** to **C** may be defined in joint coordinates as a function of h by obtaining the joint coordinates corresponding to positions **B** and **C**, $\boldsymbol{\theta}_B$ and $\boldsymbol{\theta}_C$, as:

$$\boldsymbol{\theta}_{jnt} = r\boldsymbol{\theta}_{BC} + \text{solve } (\mathbf{Expr}_{i+1}(t)), \tag{3.12}$$

where $\boldsymbol{\theta}_{BC}$ = solve (**B**)–solve (**C**) and $r = 1-h$.

The variable r changes from 1 to 0 as the motion is made. When $r = 1$ at the beginning of the motion, $\boldsymbol{\theta}_{jnt} = \boldsymbol{\theta}_B$, the initial position. When $r = 0$ at the end of the motion, $\boldsymbol{\theta}_{jnt}$ = solve ($\mathbf{Expr}_{i+1}$), the desired goal position.

The motion parameter h is defined as a polynomial function of time to provide for continuity of position, velocity, acceleration, and jerk (the third derivative of position with respect to time). A motion of constant velocity and of the above continuity specification requires the boundary conditions on h shown in Table 3.1.

Table 3.1 Boundary Conditions for h

time	$-t_{acc}$	t_{acc}	t_{seg}
h	0	t_{acc}/t_{seg}	1
$\dot{h}$	0	$1/t_{seg}$	$1/t_{seg}$
$\ddot{h}$	0	0	0
$\dddot{h}$	0	0	0

The number of boundary conditions calls for a 7th order polynomial, but the symmetry of the conditions reduces the order to six. Assume the polynomial to have the form

$$h = (\mathrm{b}_6 p^6 + \mathrm{b}_5 p^5 + \mathrm{b}_4 p^4 + \mathrm{b}_3 p^3 + \mathrm{b}_2 p^2 + \mathrm{b}_1 p + \mathrm{b}_0)\, t_{acc}/t_{seg}, \tag{3.13}$$

where p is defined linear to time as

$$p = \frac{t + t_{acc}}{2t_{acc}}. \tag{3.14}$$

Using Table 3.1, the coefficients can be easily determined so that h is defined as, for $t \le t_{acc}$,

$$h = ((2p-6)p+5)p^4\; t_{acc}/t_{seg}, \tag{3.15}$$

and, for $t > t_{acc}$,

$$h = t/t_{seg}. \tag{3.16}$$

As noted above that at position A, a discontinuity of $\theta_A-\theta_B$ in position and a discontinuity of $\dot{\theta}_A-\dot{\theta}_C$ in velocity are compensated for by a second matching polynomial. This polynomial will be defined only for $-t_{acc} \le t < t_{acc}$ to reduce the discontinuities to zero by t_{acc}. The initial acceleration and jerk of this matching polynomial are zero, as are the final acceleration and jerk. To obtain $\dot{\theta}_C$, at one sample period before the transition begins, Eq. (3.11) is evaluated to obtain $\theta_C(-t_{acc}-\tau)$, where τ is the sample period of the control system. $\dot{\theta}_C$ is then estimated by

$$\dot{\theta}_C = (\theta_C(-t_{acc})-\theta_C(-t_{acc}-\tau))/\tau. \tag{3.17}$$

The boundary conditions of the matching polynomial are shown in Table 3.2.

Table 3.2 Boundary Conditions for θ

time	$-t_{acc}$	t_{acc}
θ	$\theta_A-\theta_B$	0
$\dot{\theta}$	$\dot{\theta}_A-\dot{\theta}_C$	0
$\ddot{\theta}$	0	0
$\dddot{\theta}$	0	0

To find its coefficients, assume the polynomial of the following form:

$$\theta_{mch} = ((((a_7 p+a_6)p+a_5)p+a_4)p^3+a_1)p+a_0 \tag{3.18}$$

and, using the boundary conditions in Table 3.2, the coefficients are obtained as

$$\begin{aligned} a_0 &= \theta_A-\theta_B \\ \dot{\theta}_{AC} &= \dot{\theta}_A-\dot{\theta}_C \\ a_1 &= 2t_{acc}\dot{\theta}_{AC} \\ a_7 &= 10a_1+20a_0 \\ a_6 &= -36a_1-70a_0 \\ a_5 &= 45a_1+84a_0 \\ a_4 &= -20a_1-35a_0. \end{aligned} \tag{3.19}$$

The joint motion from position **B** to **C** is defined by Eq. (3.12) as a function of r, which in turn is a function of the path motion parameter h. The time dependence of the motion is specified by Eq. (3.13), where h is defined. This motion starts from rest, accelerates to the desired path velocity and then moves at constant velocity. The discontinuity in position and velocity between the two motions at the beginning of the path is removed by the addition of the second matching polynomial, defined by Eq. (3.18), during the accelerating portion of the path segment.

3.4.3. Cartesian Coordinate Motion

If a Cartesian motion is desired from **B** to **C**, the position Eq. (3.11) must be modified to include a drive transform $\mathbf{D}(r)$ [Paul and Zhang 1984] in **Expr**. The drive transform represents a translation **p** and rotation ψ about an axis **e** in space, both proportional to r. When the argument r is zero, representing the end of the motion, $\mathbf{D}(r)$ reduces to an identity transform. The position of the drive transform in the equation determines the frame in which the rotation is defined.

To define **D**, rewrite Eq. (3.11) with the drive transform included in the form

$$P_i\colon \mathbf{T}_n = \mathbf{L}_i\ \mathbf{D}(0)\ \mathbf{R}_i. \tag{3.20}$$

Here $\mathbf{L}_i$ represents the transform expression to the left of **D** and $\mathbf{R}_i$ represents the transform expression to the right. At the beginning of the motion, position **B** is defined by Eq. (3.12). This position is also defined with respect to position **C** in terms of $\mathbf{D}(r)$ with $r=1$.

$$\mathbf{B} = \mathbf{L}_C(-t_{acc})\ \mathbf{D}(1)\ \mathbf{R}_C(-t_{acc}), \tag{3.21}$$

which is solved for $\mathbf{D}(1)$, or

$$\mathbf{D}(1) = \mathbf{L}_C^{-1}(-t_{acc})\ \mathbf{B}\ \mathbf{R}_C^{-1}(-t_{acc}). \tag{3.22}$$

From Eq. (3.22), **p**, **e**, and ψ can be obtained and the Cartesian motion from position **B** to **C** may then be described by

$$\boldsymbol{\theta}_{car} = solve(\mathbf{L}_C(t)\ \mathbf{D}(r)\ \mathbf{R}_C(t)), \tag{3.23}$$

as r=1$-h$ varies from 1 to 0, where motion parameter h is defined in Eqs. (3.15) and (3.16). The transform $\mathbf{D}(r)$ represents a rotation of an angle $r\psi$ about a unit vector **e** and a translation $r\mathbf{p}$. The unit vector **e** is defined in terms of the first two Euler angles, ϕ and θ, as:

$$\mathbf{e} = C_\phi S_\theta \mathbf{i} + S_\phi S_\theta \mathbf{j} + C_\theta \mathbf{k}, \tag{3.24}$$

where $0 \le \theta < \pi$. C_ϕ, S_ϕ, C_θ, etc. stand for cos(ϕ), sin(ϕ), cos(θ), etc. The translation is defined to be $\mathbf{p} = rx\mathbf{i} + ry\mathbf{j} + rz\mathbf{k}$. With **p** and **e**, $\mathbf{D}(r)$ is defined by [Paul 1981]:

$$\mathbf{D}(r) = \begin{bmatrix} C_\phi^2 S_\theta^2 V_{r\psi} + C_{r\psi} & C_\phi S_\phi S_\theta^2 V_{r\psi} - C_\theta S_{r\psi} & C_\phi S_\theta C_\theta V_{r\psi} + S_\phi S_\theta S_{r\psi} & rx \\ C_\phi S_\phi S_\theta^2 V_{r\psi} + C_\theta S_{r\psi} & S_\phi^2 S_\theta^2 V_{r\psi} + C_{r\psi} & S_\phi S_\theta C_\theta V_{r\psi} - C_\phi S_\theta S_{r\psi} & ry \\ C_\phi S_\theta C_\theta V_{r\psi} - S_\phi S_\theta S_{r\psi} & S_\phi S_\theta C_\theta V_{r\psi} + C_\phi S_\theta S_{r\psi} & C_\theta^2 V_{r\psi} + C_{r\psi} & rz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.25}$$

where $C_{r\psi}$, $S_{r\psi}$, and$V_{r\psi}$, stand for *cos* $(r\psi)$, *sin* $(r\psi)$, and $(1-C_{r\psi})$, respectively.

To solve for x, y, z, ψ, ϕ, θ, define the elements of $\mathbf{L}_C^{-1}(-t_{acc})\ \mathbf{B}\ \mathbf{R}_C^{-1}(-t_{acc})$ from Eq. (3.21) to be

$$\mathbf{L}_C^{-1}(-t_{acc})\ \mathbf{B}\ \mathbf{R}_C^{-1}(-t_{acc}) = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.26}$$

One then solves the matrix equation with $\mathbf{D}(1)$ on one side and the right hand side of Eq. (3.26) on the other, directly for the parameters x, y, z, ψ, ϕ, θ.

$$\begin{bmatrix} C_\phi^2 S_\theta^2 V_\psi + C_\psi & C_\phi S_\phi S_\theta^2 V_\psi - C_\theta S_\psi & C_\phi S_\theta C_\theta V_\psi + S_\phi S_\theta S_\psi & x \\ C_\phi S_\phi S_\theta^2 V_\psi + C_\theta S_\psi & S_\phi^2 S_\theta^2 V_\psi + C_\psi & S_\phi S_\theta C_\theta V_\psi - C_\phi S_\theta S_\psi & y \\ C_\phi S_\theta C_\theta V_\psi - S_\phi S_\theta S_\psi & S_\phi S_\theta C_\theta V_\psi + C_\phi S_\theta S_\psi & C_\theta^2 V_\psi + C_\psi & z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

As matrix equality implies element-by-element equality, the following may be obtained directly from Eq. (3.27):

$$x = p_x,\ y = p_y,\ \text{and } z = p_z. \tag{3.28}$$

Equating the sum of the diagonal elements from Eq. (3.27),

$$1 + 2C_\psi = n_x + o_y + a_z, \tag{3.29}$$

and thus

$$C_\psi = \tfrac{1}{2}(n_x + o_y + a_z - 1). \tag{3.30}$$

Equating the difference of the off-diagonal pairs of the elements of Eq. (3.27),

$$4S_\psi^2 = (n_y - o_x)^2 + (a_x - n_z)^2 + (o_z - a_y)^2 \tag{3.31}$$

and, as the rotation angle ψ is always less than π,

$$S_\psi = +\tfrac{1}{2}\sqrt{(n_y - o_x)^2 + (a_x - n_z)^2 + (o_z - a_y)^2}. \tag{3.32}$$

Finally, an expression for ψ from Eqs. (3.30) and (3.32) is obtained as

$$\psi = \tan^{-1} \frac{+\sqrt{(n_y - o_x)^2 + (a_x - n_z)^2 + (o_z - a_y)^2}}{(n_x + o_y + a_z - 1)}. \tag{3.33}$$

Equating off-diagonal pairs from Eq. (3.27) ,

$$\begin{aligned} 2C_\theta S_\psi &= n_y - o_x, \\ 2S_\phi S_\theta S_\psi &= a_x - n_z, \\ 2C_\phi S_\theta S_\psi &= o_z - a_y. \end{aligned} \tag{3.34}$$

Squaring and adding the 2nd and 3rd Equations in (3.34),

$$4S_\theta^2 S_\psi^2 = (a_x - n_z)^2 + (o_z - a_y)^2. \tag{3.35}$$

As $0 \le \theta < \pi$, S_θ is obtained as

$$S_\theta = +\frac{1}{2S_\psi}\sqrt{(a_x - n_z)^2 + (o_z - a_y)^2} \tag{3.36}$$

C_θ is obtained directly from Eq. (3.34) as

$$C_\theta = \frac{1}{2S_\psi}(n_y - o_x), \tag{3.37}$$

and thus

$$\theta = \tan^{-1}\frac{+\sqrt{(a_x - n_z)^2 + (o_z - a_y)^2}}{(n_y - o_x)}. \tag{3.38}$$

ϕ can be solved for from 2nd and 3rd Equations of (3.34):

$$\begin{aligned} S_\phi &= \frac{1}{2S_\theta S_\psi}(a_x - n_z) \\ C_\phi &= \frac{1}{2S_\theta S_\psi}(o_z - a_y). \end{aligned} \tag{3.39}$$

As $0 \le \theta < \pi$ and $0 \le \psi < \pi$, ϕ is obtained uniquely as

$$\phi = \tan^{-1}\frac{(a_x - n_z)}{(o_z - a_y)} \tag{3.40}$$

Note that the definitions for the angular drive parameters break down as the angle of rotation ψ approaches π. This lack of stability of the solution reflects reality, as reorientations of approximately π are extremely unstable with the direction or rotation of the manipulator wrist reversing as the angle moves through π. It is therefore reasonable to restrict reorientations to less than approximately 0.8π. With this restriction, the definitions of the drive parameters are valid. Rotations of greater than 0.8π can be executed by a sequence of rotations, each of less than 0.8π.

If the manipulator is already in motion when motion from **B** to **C** starts, a matching polynomial must be added to the trajectory to remove any discontinuity. Since Equation (3.23) is derived independently of the previous motion, the matching polynomial given by Eq. (3.18) is calculated with the same boundary conditions in Table 3.1.

3.5. Discussion

The proposed method uses a distinct strategy to handle transitions all in joint space, so that the four possible cases are dealt with by one method. This is made possible by the property that the matching polynomial is independent from the next mode of motion. As summarized in Table 3.3, transition may start at $t=t_{seg}-t_{acc}-\tau$ from joint or Cartesian motion to joint or Cartesian motion. This greatly simplies the programming and understanding of the method, as demonstrated by the trajectory generating program in Appendix C.

The computational complexity of the method varies from one sampling period to another. If a uniform sampling period is used, the achievable sampling rate depends only on the worst period in terms of time. In the above trajectory generator, the worst case occurs in Cartesian motion at beginning of a transition to a Cartesian motion regardless of the mode of the current motion, when the current set-point, the next drive transform, as well as the coefficients of the matching polynomial are all computed. This corresponds to one $\mathbf{T}_6$ derivation, two inverse kinematic solutions, and a drive transform calculation. This contrasts with [Paul 1981], in which transition to a Cartesian motion is computed in Cartesian space. If the current motion is a joint motion, it must be converted to a Cartesian motion first, requiring a direct kinematic solution. In addition, two drive transforms, coefficients of the matching polynomial, and an inverse kinematic solution are also computed. More importantly, since the transition takes place in Cartesian space, the computation of the matching polynomial with vector variables cannot be distributed, as opposed to to a scalar polynomial in the proposed

Table 3.3 Motion Control Summary

Time	Joint Motion	Cartesian Motion
$t_{acc}<t<t_{seg}-t_{acc}-\tau$	$h = t/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = r\boldsymbol{\theta}_{BC}+\text{solve }(\mathbf{EXPR}_{i-1}(h,t))$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$	$h = t/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = \text{solve }(\mathbf{L}_{i-1}(h,t)\ \mathbf{D}(r)\ \mathbf{R}_{i-1}(h,t))$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$
$t = t_{seg}-t_{acc}-\tau$	$h = t/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = r\boldsymbol{\theta}_{BC}+\text{solve }(\mathbf{EXPR}_{i-1}(h,t))$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$ $\boldsymbol{\theta}_{TMP} = \text{solve }(\mathbf{EXPR}_{i}(0,t))$	$h = t/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = \text{solve }(\mathbf{L}_{i-1}(h,t)\ \mathbf{D}(r)\ \mathbf{R}_{i-1}(h,t))$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$ $\boldsymbol{\theta}_{TMP} = \text{solve }(\mathbf{EXPR}_{i}(0,t))$
$t = t_{seg}-t_{acc}$	$\boldsymbol{\theta} = \boldsymbol{\theta}+\dot{\boldsymbol{\theta}}\tau$ $\boldsymbol{\theta}_B = \text{solve }(\mathbf{EXPR}_{i-1}(1,t))$ $\boldsymbol{\theta}_C = \text{solve }(\mathbf{EXPR}_{i}(0,t))$ $\mathbf{a}_0 = \boldsymbol{\theta}-\boldsymbol{\theta}_B$ $\boldsymbol{\theta}_{BC} = \boldsymbol{\theta}_B-\boldsymbol{\theta}_C$ $\dot{\boldsymbol{\theta}}_C = (\boldsymbol{\theta}_C-\boldsymbol{\theta}_{TMP})/\tau$ $\dot{\boldsymbol{\theta}}_{AC} = \dot{\boldsymbol{\theta}}-\dot{\boldsymbol{\theta}}_C$ $\mathbf{a}_1 = 2t_{acc}\dot{\boldsymbol{\theta}}_{AC}$ $\mathbf{a}_7 = 10\mathbf{a}_1+20\mathbf{a}_0$ $\mathbf{a}_6 = -36\mathbf{a}_1-70\mathbf{a}_0$ $\mathbf{a}_5 = 45\mathbf{a}_1+84\mathbf{a}_0$ $\mathbf{a}_4 = -20\mathbf{a}_1-35\mathbf{a}_0$ $t = -t_{acc}+\tau$	$\boldsymbol{\theta} = \boldsymbol{\theta}+\dot{\boldsymbol{\theta}}\tau$ $\mathbf{B} = \mathbf{EXPR}_{i-1}(1,t)$ $\boldsymbol{\theta}_C = \text{solve }(\mathbf{EXPR}_{i}(0,t))$ $\mathbf{a}_0 = \boldsymbol{\theta}-\text{solve }(\mathbf{B})$ Solve for $x, y, z, \psi, \phi, \theta$ from $\mathbf{D}(1) = \mathbf{L}_i^{-1}(0,t)\ \mathbf{B}\ \mathbf{R}_i^{-1}(0,t))$ $\dot{\boldsymbol{\theta}}_C = (\boldsymbol{\theta}_C-\boldsymbol{\theta}_{TMP})/\tau$ $\dot{\boldsymbol{\theta}}_{AC} = \dot{\boldsymbol{\theta}}-\dot{\boldsymbol{\theta}}_C$ $\mathbf{a}_1 = 2t_{acc}\dot{\boldsymbol{\theta}}_{AC}$ $\mathbf{a}_7 = 10\mathbf{a}_1+20\mathbf{a}_0$ $\mathbf{a}_6 = -36\mathbf{a}_1-70\mathbf{a}_0$ $\mathbf{a}_5 = 45\mathbf{a}_1+84\mathbf{a}_0$ $\mathbf{a}_4 = -20\mathbf{a}_1-35\mathbf{a}_0$ $t = -t_{acc}+\tau$
$-t_{acc}<t\leq t_{acc}$	$p = (t+t_{acc})/(2t_{acc})$ $h = ((2p-6)p+5)p^4 * t_{acc}/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = r\boldsymbol{\theta}_{BC}+\text{solve }(\mathbf{EXPR}_{i}(h,t))$ $\quad +((((\mathbf{a}_7 p+\mathbf{a}_6)p+\mathbf{a}_5)p+\mathbf{a}_4)p^3+\mathbf{a}_1)p+\mathbf{a}_0$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$	$p = (t+t_{acc})/(2t_{acc})$ $h = ((2p-6)p+5)p^4 * t_{acc}/t_{seg}$ $r = 1-h$ $\boldsymbol{\theta} = \text{solve }(\mathbf{L}_{i}(h,t)\ \mathbf{D}(r)\ \mathbf{R}_{i}(h,t))$ $\quad +((((\mathbf{a}_7 p+\mathbf{a}_6)p+\mathbf{a}_5)p+\mathbf{a}_4)p^3+\mathbf{a}_1)p+\mathbf{a}_0$ $\dot{\boldsymbol{\theta}} = d\boldsymbol{\theta}/dt$

method above, where computation of the matching polynomial can be distributed to the joints.

In a manner similar to [Paul 1981], sensor-guided motions are implemented by defining *ref* transforms in position equations and compliant motion is achieved by adding a **COMPLY** matrix at the appropriate location in the position equation. The terminal cases from rest to motion or from motion to rest are not explicit in the above algorithm. However, both can be considered as special cases of the general scheme in Figure 3.1. A motion from rest requires an initialization process which sets $\boldsymbol{\theta}$ to current position and velocity to zero and computes a $\boldsymbol{\theta}_{TMP}$ from the destination position as if it were one sampling period before the transition at $-t_{acc}$. A motion to rest can be considered as one with the next destination position the same as the current one so that a transition to the current destination is performed with the segment time set to t_{acc} Since there is no position nor velocity difference between the current and the next positions, the manipulator automatically comes to a stop.

When a move to a position is started, the manipulator will not pass through that position because a transition to a trajectory of the next goal position occurs before the first goal position is reached. The manipulator must come to a stop at each intermediate position along a path in order to reach each position exactly. It is also possible to compute deviation of the actual trajectory from the goal [Paul 1981].

As examples, the above trajectory generator has been implemented for motion control of PUMA 250 manipulator [Unimation 1980]. Its kinematics can be found in Appendix A. Figure 3.3 shows a joint motion trajectory of joint 1, its position, velocity, acceleration, and jerk plots. Figure 3.4 shows the same motion with longer t_{acc}. Figure 3.5 shows the trajectory of joint one in executing a Cartesian motion. Notice in this case, velocity is no longer linear in Joint space.

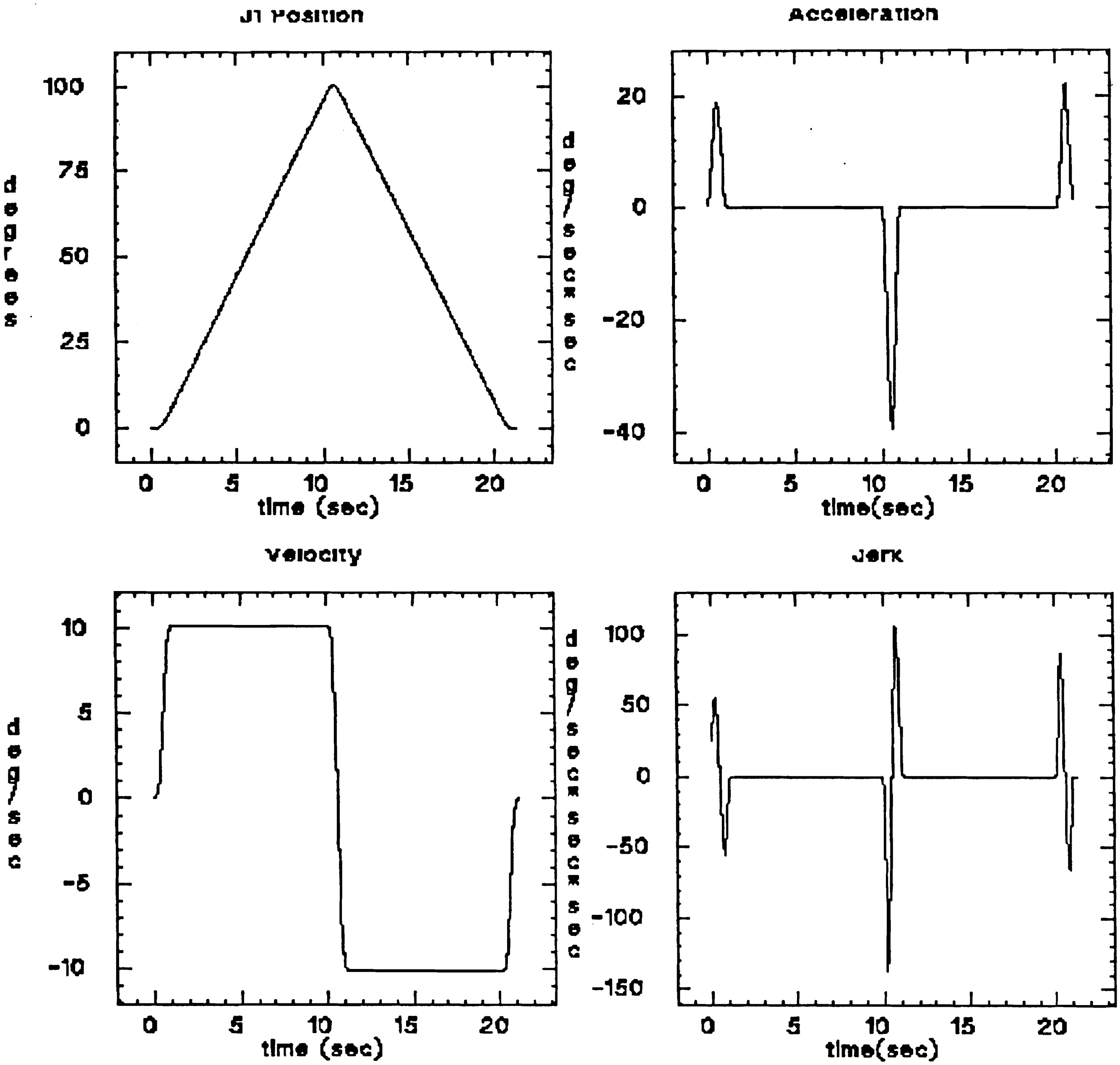


Figure 3.3. Trajectories of Joint One in Joint Mode

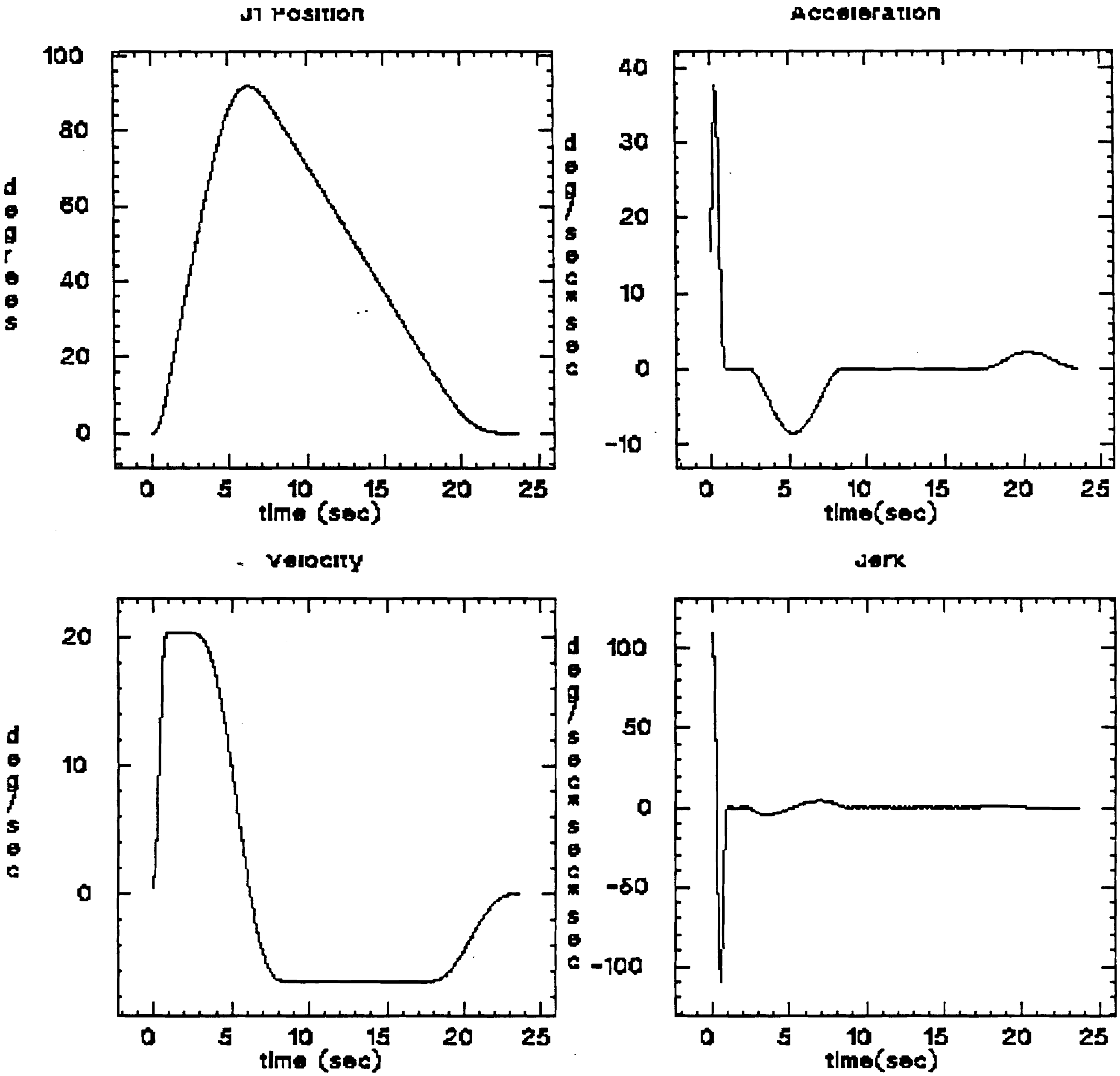
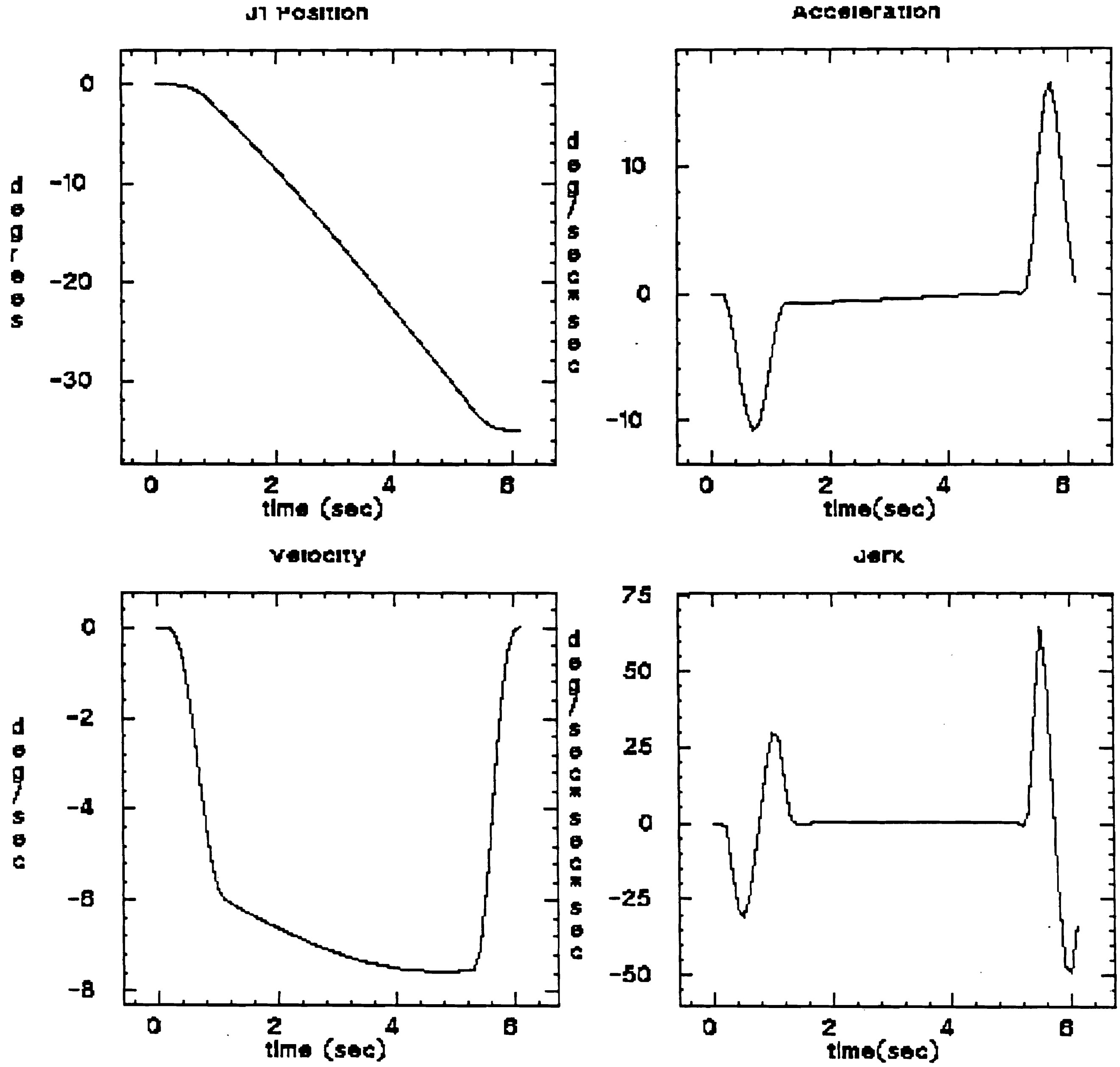


Figure 3.4. Joint Mode Trajectories with Longer t_{acc}


J1 Position

Acceleration

Velocity

Jerk

Figure 3.5. Trajectories of Joint One in Cartesian Mode

4. Conclusion

This section has discussed the specification of motion for a robot manipulator, reviewed different methods which satisfy the motion specification, and introduced a unified approach to motion trajectory generation. The approach supports both Cartesian mode of motion and joint mode of motion, either of which can be functionally defined, and treats the transition in a simple and efficient manner. The Table 3.3 summerizes the approach.

CHAPTER IV

CALCULATION OF MANIPULATOR DYNAMICS

1. Introduction

The problem of the dynamics of a robot manipulator is given the Cartesian position, velocity, and desired acceleration of the end of the manipulator ($\mathbf{x}$, $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$), find the joint torque vector $\boldsymbol{\tau}$ necessary to generate the desired acceleration. The problem is important for a number of reasons. The control system that does not take the dynamics into consideration will always have its trajectory deviating from the desired trajectory due to joint torque errors. When a manipulator is to apply forces, the dynamics also plays an essential part in the accuracy of the force application.

Even though the theoretical problems in manipulator dynamics have been solved [Bejczy and Lee 1983, Hughes 1977, and Luh, Walker and Paul 1980], the question of how to apply the theories to manipulator control still remains to be answered. This chapter first examines the two basic formulations of manipulator dynamics, the Lagrangian formulation and the Newton-Euler formulation, and establishes the superiority of the Lagrangian method in the context of manipulator control. Based on significance analysis, the centripetal and Coriolis coefficients in the symbolic Lagrangian equations are ignored and the remaining coefficients derived, as an example, for PUMA 560 manipulator. An experimental method is then proposed and used to determine the constants in the simplified dynamics equations.

2. Formulations of Dynamics Equations

There have been efforts in recent years on the study of the manipulator dynamics and much progress has been made [Khan 1969, Bejczy 1974, Paul 1981, Hollerbach 1980, Hughes 1977, and Luh, Walker and Paul 1980]. Two major approaches in terms of formulation of the dynamics equations are the Newton-Euler method [Luh, Walker and Paul 1980] and the Lagrangian formulation[Bejczy 1974]. As in the case of solving the linear system $\mathbf{x} = A\mathbf{y}$, given an $\mathbf{x}$, there are two ways of obtaining $\mathbf{y}$: one can either use Gaussian elimination to obtain components of $\mathbf{y}$ one by one or find the inverse of matrix $\mathbf{A}$ to obtain $\mathbf{y}$ all at once. The Newton-Euler method solves the problem recursively to find joint torques one by one whereas the Lagrangian method solves it by closed-form equations. Newton-Euler method requires less computation per iteration of the solution than does the Lagrangian method. However, once the closed-form equations are obtained in Lagrangian method, they remain valid provided the manipulator configuration remains unchanged.

The Lagrangian formulation was first developed to compute closed-form manipulator dynamics [Uicker 1966 and Kahn 1969]. The formulation is based on the Lagrangian equation

$$\tau_i = \frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial \dot{\theta}_i}\right) - \frac{\partial \mathrm{L}}{\partial \theta_i} \qquad i = 1, \cdots, n \tag{4.1}$$

where $\mathrm{L} = \mathrm{K} - \mathrm{P}$ is the Lagrangian and K and P are kinetic energy and potential energy of the manipulator. Using the expressions for K and P in terms of manipulator parameters, the dynamics equation for joint i is obtained as

$$\tau_i = \sum_{j=1}^{6} \mathrm{D}_{ij}\ddot{\theta}_j + Ia_i\ddot{\theta}_i + \sum_{j=1}^{6}\sum_{k=1}^{6} \mathrm{D}_{ijk}\dot{\theta}_j\dot{\theta}_k + \mathrm{D}_i \tag{4.2}$$

where

$$\mathrm{D}_{ij} = \sum_{p=max\ i,j}^{6} \mathrm{Trace}\left[\frac{\partial \mathbf{T}_p}{\partial \theta_j} J_p \frac{\partial \mathbf{T}_p{}^{\mathrm{T}}}{\partial \theta_i}\right] \tag{4.3}$$

$$D_{ijk} = \sum_{p=max\ i,j,k}^{6} \text{Trace}\left[\frac{\partial^2 \mathbf{T}_p}{\partial\theta_j \partial\theta_k} J_p \frac{\partial \mathbf{T}_p{}^{\mathrm{T}}}{\partial\theta_i}\right] \tag{4.4}$$

$$D_i = \sum_{p=i}^{6} -m_p g^{\mathrm{T}} \frac{\partial \mathbf{T}_p}{\partial\theta_i} {}^{p}\bar{\mathbf{r}}_p \tag{4.5}$$

In the above equations, $\mathbf{T}_p = \mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_p$, m_i is the mass of link i, ${}^{i}\bar{\mathbf{r}}_i$ is the center of m_i with respect to the base of link i, $\mathbf{J}_i$ is the pseudo inertia matrix of link i, and Ia_i is the actuator inertia of link i.

This formulation explicitly expresses the dynamics in terms of gravity loading D_i, effective inertia D_{ii} and coupling inertias D_{ij} where $i \neq j$, and Coriolis and centripetal coefficients D_{ijk}. The evaluation of the dynamic terms requires tens of thousands of arithmetic operations [Brady, *et al.* 1982] for each update. One common practice to reduce the complexity of Lagrangian dynamics equations is to derive the dynamics terms symbolically. The method of symbolic derivation of dynamics model was first introduced by Bejczy [1974] [Bejczy and Lee 1983]. The dynamics model of a manipulator carrying a load was also derived by Izaguirre [1984]. Many efficient procedures for generating dynamics models of robot manipulators have also been devised and automatic systems built [Murray 1983, and Cesareo 1983].

The Newton-Euler method is based on Newton's law of linear motion and Euler's equation of angular motion that for a rigid body i, a linear movement or an angular rotation requires force $\mathbf{f}_i = m_i\ddot{\mathbf{r}}$ or net torque $\tau_i = \mathbf{J}_i\dot{\omega}_i + \omega_i \times \mathbf{J}_i\omega_i$, where $\mathbf{r}$ is the center of mass, J is the pseudo inertial matrix and ω_i is the angular velocity.

The method consists a set of forward equations, which propagate velocities and accelerations of the joints from the base to the end of the manipulator, and a set of backward equations, which propagate backwards joint torques or forces due to the velocities and accelerations from the end to the base. If all joints are revolute, for example, the forward equations have the form

$$\begin{aligned}
&\mathbf{A}_{i+1}^{0}\underline{\omega}_{i+1} = \mathbf{A}_{i+1}^{i}(\mathbf{A}_{i+1}^{0}\underline{\omega}_{i} + \mathbf{z}_0\dot{\theta}_{i+1})\\
&\mathbf{A}_{i+1}^{0}\underline{\dot{\omega}}_{i+1} = \mathbf{A}_{i+1}^{i}[\mathbf{A}_{i}^{0}\underline{\dot{\omega}}_{i} + \mathbf{z}_0\ddot{\theta}_{i+1} + (\mathbf{A}_{i}^{0}\underline{\omega}_{i})\times(\mathbf{z}_0\dot{\theta}_{i+1})]\\
&\mathbf{z}_0 = [0\ 0\ 1]^{\mathrm{T}}\\
&\mathbf{A}_{i+1}^{0}\mathbf{v}_{i+1} = (\mathbf{A}_{i+1}^{0}\underline{\omega}_{i+1})\times(\mathbf{A}_{i+1}^{0}\mathbf{p}_{i+1}^{*}) + \mathbf{A}_{i+1}^{i}(\mathbf{A}_{i}^{0}\mathbf{v}_{i})\\
&\mathbf{A}_{i+1}^{0}\dot{\mathbf{v}}_{i+1} = (\mathbf{A}_{i+1}^{0}\underline{\dot{\omega}}_{i+1})\times(\mathbf{A}_{i+1}^{0}\mathbf{p}_{i+1}^{*}) + (\mathbf{A}_{i+1}^{0}\underline{\omega}_{i+1})\times[(\mathbf{A}_{i+1}^{0}\underline{\omega}_{i+1})\times(\mathbf{A}_{i+1}^{0}\mathbf{p}_{i+1}^{*})]\\
&\qquad\qquad + \mathbf{A}_{i+1}^{i}(\mathbf{A}_{i}^{0}\dot{\mathbf{v}}_{i})
\end{aligned}\qquad\text{(4.6}$$

for $i = 0,\ 1,\ \cdots,\ n-1$.

Let

$$\begin{aligned}
&\mathbf{A}_{i}^{0}\mathbf{F}_{i} = m_i\mathbf{A}_{i}^{0}\dot{\hat{\mathbf{v}}}_{i}\\
&\mathbf{A}_{i}^{0}\mathbf{N}_{i} = (\mathbf{A}_{i}^{0}\mathbf{J}_{i}\mathbf{A}_{0}^{i})(\mathbf{A}_{i}^{0})(\underline{\dot{\omega}}_{i}) + (\mathbf{A}_{i}^{0}\underline{\omega}_{i}) \times [(\mathbf{A}_{i}^{0}\mathbf{J}_{i}\mathbf{A}_{0}^{i})(\mathbf{A}_{i}^{0}\underline{\omega}_{i})]
\end{aligned}\qquad(4.7)$$

then the backward equations will be

$$\begin{aligned}
&\mathbf{A}_{i}^{0}\mathbf{f}_{i} = \mathbf{A}_{i}^{i+1}(\mathbf{A}_{i+1}^{0}\mathbf{f}_{i+1}) + m_i\mathbf{A}_{i}^{0}\dot{\mathbf{v}}_{i}\\
&\mathbf{A}_{i}^{0}\mathbf{n}_{i} = \mathbf{A}_{i}^{i+1}[\mathbf{A}_{i+1}^{0}\mathbf{n}_{i+1} + (\mathbf{A}_{i+1}^{0}\mathbf{p}_{i}^{*})\times(\mathbf{A}_{i+1}^{0}\mathbf{f}_{i+1})] + (\mathbf{A}_{i}^{0}\mathbf{p}_{i}^{*} + \mathbf{A}_{i}^{0}\hat{\mathbf{s}}_{i})\times(\mathbf{A}_{i}^{0}\mathbf{F}_{i})\\
&\qquad + \mathbf{A}_{i}^{0}\mathbf{N}_{i}\\
&\tau_i = (\mathbf{A}_{i}^{0}\mathbf{n}_{i})^{T}(\mathbf{A}_{i}^{i+1}\mathbf{z}_0)
\end{aligned}\qquad(4.8)$$

for $i = n-1,\ \cdots,\ 1,\ 0$.

Newton-Euler method is much more efficient computationally than the Lagrangian formulation because of the recursive definition. the rotation is represented as three angles as opposed to a rotational 3x3 matrix used by Lagrangian method. The formulation, however, does not explicitly generate the different dynamic terms.

It has been shown that the two formulations are equivalent to each other in the sense that algorithms exist to compute Lagrangian equations recursively as a backward recursion and a forward recursion and, further, than the Newton-Euler method with proper rotation representation can be expressed in closed-form, of which computational complexity also becomes equal to that of Lagrangian formulation [Silver 1982].

3. Comparisons of the Two Formulations

Both the Newton-Euler and the Lagrangian formulation involve too many numerical operations in their present forms to be applied to real-time control. Similar to the situation of manipulator trajectory generation, the requirement on the rate at which dynamics is supplied to the joints is determined by the control system, which, in addition to computing the manipulator set-points, has to compute dynamics as well. Therefore, these methods must be simplified to reduce the computations necessary to update the dynamics.

When the dynamics is calculated by a parallel computer, the Lagrangian formulation is preferred, with each joint computing its own dynamics in parallel with others. The Newton-Euler method, on the other hand, is inherently a serial process. Although there have been attempts [Luh and Lin 1983, Nigam and Lee 1985, and Kasahara and Narita 1985] to parallelize the Newton-Euler equations, the procedure is not generally applicable and resulting systems are extremely complex and require heavy variable sharing and data interaction among processes, making the implementation difficult.

When considered for real-time control of robot manipulators, the Newton-Euler method does not really provide a feasible solution, since the formulation of the equations requires that they be updated at the rate of the joint servos, regardless of whether the dynamics of the system changes at that rate. Given the present computer technology, system capable of performing such a throughput for a robot manipulator cannot be economically justified. In contrast, the Lagrangian equations can be computed independently of the manipulator joint servos when the dynamic terms are derived symbolically. This makes it possible to update the manipulator dynamics not at the rate of the joint servos but at the rate of manipulator configuration changes, of which the dynamics equations are functions. It is also possible in Lagrangian equations to simplify the dynamics terms based on the significant analysis once they are derived.

4. Simplification of Lagrangian Dynamics Equations

Bejczy first noticed the disparity of the roles that different dynamics terms play in the dynamics equations [Bejczy 1974] and Paul extended the idea to the elimination of the insignificant dynamics terms and expressions within terms when using the equations for manipulator control [Paul 1981, and Paul *et al.* 1983]. The complete elimination of the velocity dependent terms D_{ijk} has been a subject of much controversy. It has been shown [Brady, *et al.* 1982] that there are situations where centripetal and Coriolis forces dominate the inertial forces. In general, however, the manipulator joints experience high velocities only during the gross motions when the accuracy of the control is not critical. During the fine motions when the control accuracy is important, joints move with high accelerations and very low velocities so that the gravitational and inertial forces become dominant and, therefore, velocity dependent forces can be justly ignored.

Based on the above argument it can be concluded that Coriolis forces play a much less significant role in manipulator dynamics than the inertial and gravitational terms and consequently do not justify computation required when the triple summation in Eq. (4.2) is to be computed. If only limited computing power is available and approximation must be made, centripetal and Coriolis forces should be ignored in dynamics computation.

Further simplifications can be made in the expressions of the dynamics terms. These symbolic terms are functions of manipulator link parameters such as link masses, center of masses, and radii of gyration. One can examine the relative significance within an expression and ignore the less significant terms. For example, when the x component of a center of mass is far less than the y component and if they are to be added, one can approximate the result by the y component. Such simplified Dynamics equations based on Lagrangian method for PUMA 560 manipulator has been derived for the gravity loadings and effective inertias and one coupling inertia term D_{23} in [Paul, Ma and Zhang 1983]. The detailed results and center of masses and radii of gyration are also given in Appendix B. The final symbolic expressions are listed below.

4.1. Gravity Loadings

When the symbolic expressions are obtained, all matrix operations are replaced by scalar operations. Gravity loadings for PUMA 560 manipulator are obtained as follows:

$$\begin{aligned}
D_6 &= 0 \\
D_5 &= c^{l}{}_{50}(C_{23}S_5 + S_{23}C_4C_5) \\
D_4 &= -c^{l}{}_{50}S_{23}S_4S_5 \\
D_3 &= c^{l}{}_{31}C_{23} + c^{l}{}_{32}S_{23} \\
D_2 &= c^{l}{}_{32}S_{23} + c^{l}{}_{31}C_{23} + c^{l}{}_{21}C_2 \\
D_1 &= 0
\end{aligned} \tag{4.9}$$

where the symbolic expressions for the constants c_{ij} dependent on link parameters can be found in Appendix B.

4.2. Effective Inertias

The symbolic effective inertias D_{ii} for PUMA 560 manipulator are obtained as follows:

$$\begin{aligned}
D_{66} &= b^{l}{}_{60} \\
D_{55} &= b^{l}{}_{50} \\
D_{44} &= b_{40} + b^{l}{}_{41}S^2{}_5 \\
D_{33} &= b^{l}{}_{30} + b^{l}{}_{31}C_5 \\
D_{22} &= b^{l}{}_{20} + b^{l}{}_{25}C_3 + b^{l}{}_{26}S_3 \\
D_{11} &= b^{l}{}_{10} + b^{l}{}_{11}C^2{}_2 + b^{l}{}_{12}C^2{}_{23} + b^{l}{}_{13}C_2C_{23} + b^{l}{}_{14}C_2S_{23} + b^{l}{}_{15}S_{23}C_{23}
\end{aligned} \tag{4.11}$$

where the symbolic expressions for constants b_{ij} dependent on link parameters can be found in Appendix B.

4.3. Coupling Inertias

One coupling inertial term, D_{23} more significant than any other, has been derived.

$$D_{23} = b^{l}_{230} + b^{l}_{231} C_3 + b^{l}_{232} S_3 \tag{4.12}$$

The expressions for b_{ijk} can be found in Appendix B and a complete set of inertia terms for PUMA 560 manipulator can be found in [Bejczy and Lee 1983].

5. Determination of the Dynamics Constants

The symbolic expressions for the dynamics consists of variables, which are functions of sines and cosines of joint positions, and constants like b_{ij} and c_{ij}, which depend on the manipulator link parameters such as link mass, center of mass, and radii of gyrations. When the dynamics equations are used in the control system of a manipulator, the values of the constants must be determined. One may actually take measurements of the links or read engineering drawings of the manipulator to obtain the dimensions of centers of mass and radius of gyration of each link, calculate the links masses by the measurements and the density of the materials the links are made of, and compute the dynamics constants using the symbolic expressions. Although values of the link parameters can be accurately calculated from the measurements and the drawings, the process is tedious and the calculated values can sometimes be in error.

An alternative to obtain the constants is an empirical approach. By actually running the manipulator, one observes joint torques necessary to generate the motion while the manipulator moves along a trajectory with known motion parameters. Since the joint torque is directly related to the constants by the dynamics equations and all intermediate joints positions as well as their sines and cosines are known, a set of equations linear to the constants can be established from the readings of joint torque and position and used to solve for the constants in the dynamics equations. If *n* constants are to be determined, at least *n* simultaneous and independent equations are necessary to uniquely determine the constants. An overconstrained simultaneous system of equations also can be used to obtain the best estimate of the constants. This method takes the nonlinearity of the manipulator into account and can be repeatedly carried out until optimal results are reached.

5.1. Joint Torque Calibration

To determine the dynamics constants experimentally, it is important to know the joint torques of all the joints at any time instant. This can be achieved by using force sensors at the joints. For a revolute joint, its force sensor records the joint torque readings; for a prismatic joint, its force sensor records the joint force readings. While the form of joint sensors may vary, if the manipulator joints are actuated by electric motors, joint motor currents provide a direct measurement on the force or torque being exerted by joints. Figure 4.1 shows a typical relationship between a joint motor current and joint output torque.

The output torque is approximately linear to the motor current except for the offset at the origin and a diverging curvature on both curves, which correspond to the two directions of motion. The offset at the origin is caused by static friction that the joint must overcome before any motion at the joint can result. The diverging characteristic explains the load dependent nature of joint friction, which increases nonlinearly with the increase in load. In practice, however, the functional relationship between joint torque and current, which is complex as well as difficult to determine, is usually approximated by a linear relationship. In this case, four quantities will fully describe each joint and the process of computing torque from current becomes a simple and efficient linear mapping. Let A_{pi} and B_{pi} be the slope and offset of the relationship for

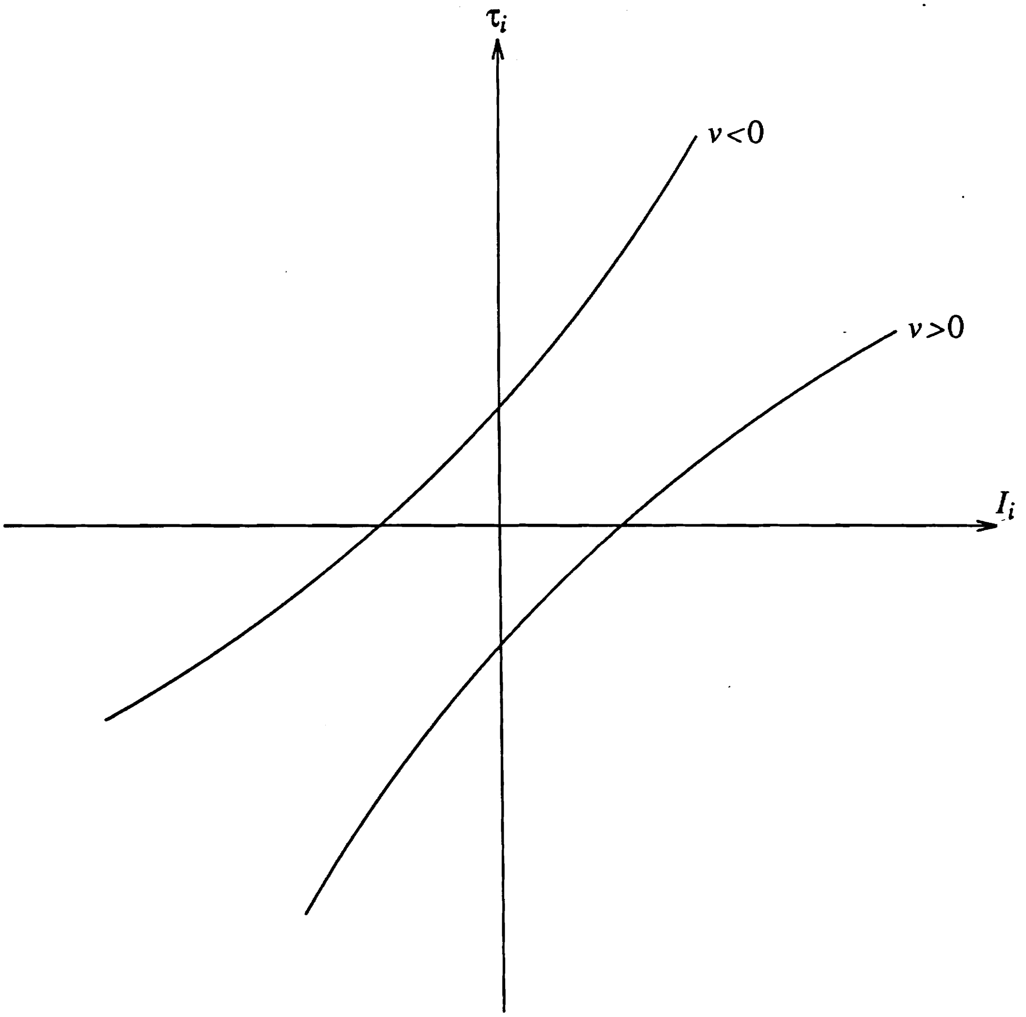


Figure 4.1. Joint Torque versus Motor Current

positive velocity and let A_{ni} and B_{ni} be the slope and offset for the negative velocity direction, the following equations represent the torque/current calibration for joint i:

$$\begin{aligned} \tau_i &= A_{pi} I_i + B_{pi} \\ \tau_i &= A_{ni} I_i + B_{ni} \end{aligned} \tag{4.13}$$

where τ_i refers to the torque value and I_i the motor current of joint i.

As a demonstration, the above method has been performed on the PUMA 560 manipulator. The measurements of output torques versus joint motor currents for each one of the six joints are illustrated in Figure 4.2. Linear approximations as described by Eq. (4.12) have also been obtained for PUMA 560 manipulator and the results are listed in Table 4.1.

Table 4.1 Joint Torque Calibration of PUMA 560

Joint	A_p	B_p	A_n	B_n
1	9.06	-2017.	8.64	1800.
2	13.44	-1088.	14.11	1280.
3	7.28	-642.	7.02	671.
4	1.72	-116.	1.77	111.
5	1.46	-101.	1.45	146.
6	1.76	-96.	1.76	88.

In Table 4.1, the slope As are in $oz{-}in/dac$ and Bs in $oz{-}in$. The unit *dac* stands for digital-to-analog conversion, in which PUMA controller sends currents to joint motors. When one reads current values from the joint motors, another unit, *adc* for analog-to-digital conversion, is usually used. In this work, the two units are related linearly by

$$I_{adc} = A_c I_{dac} + B_c \tag{4.14}$$

where, for the PUMA controller in the experiment, $A_c = 5.74$ and $B_c = -61.30$, for all joints. Table 4.1 then serves as a basic relationship from which the joint output torques are derived to determine the dynamics constants.

5.2. Determination of Gravity Loadings

To obtain gravitational constants c_{ij} in Eq. (4.2) from the knowledge of joint torques τ_i, the effects due to other dynamics terms must be eliminated so that the joint becomes a function of gravity loading. One simple method to achieve this is to move only the joint of interest and have other joints remain stationary. Under these conditions, the velocity and acceleration dependent terms disappear from the Eq. (4.2) and

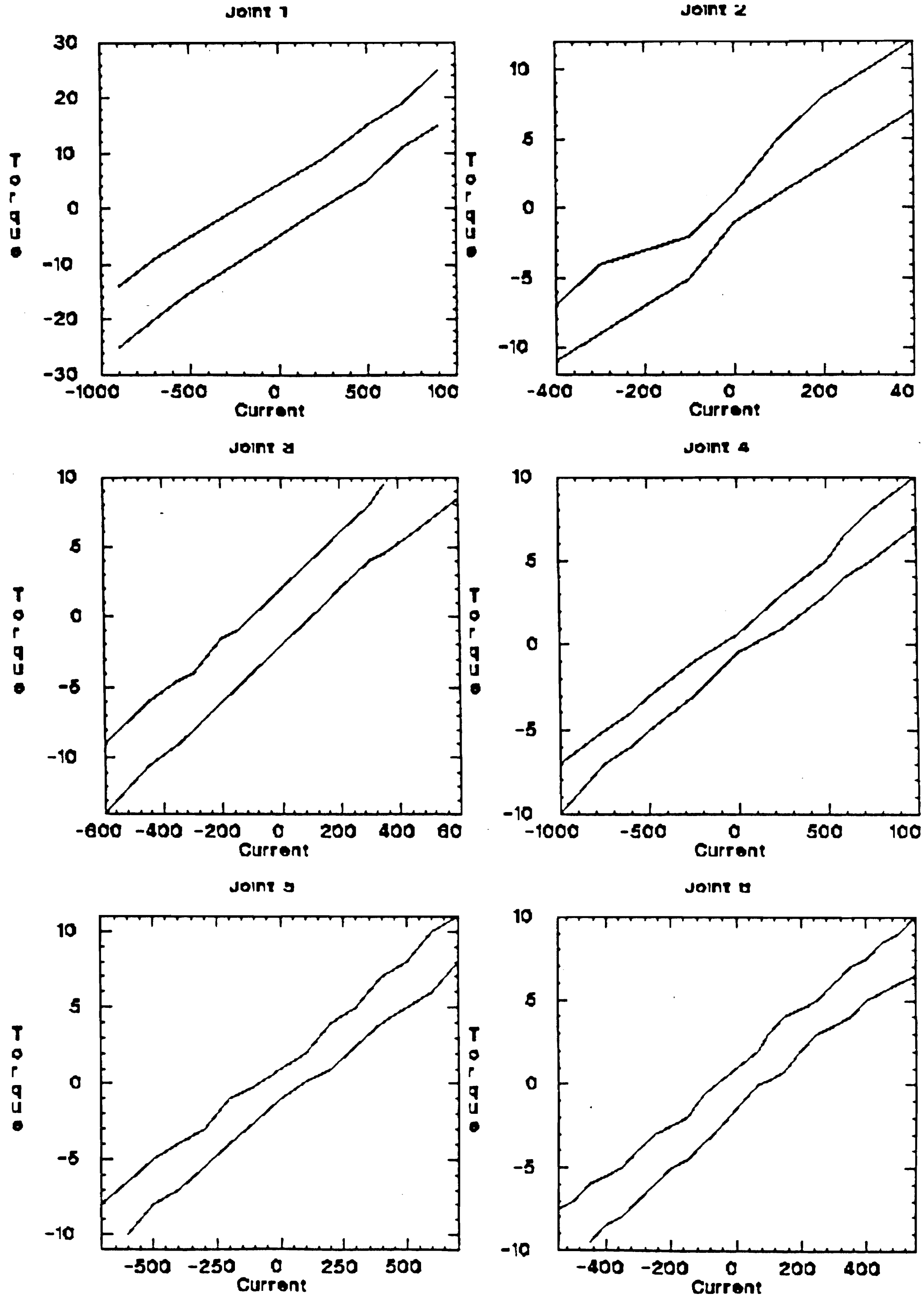


Figure 4.2. Joint Torque Calibration for the PUMA 560

the following equation results:

$$\tau_i = \mathrm{D}_i \tag{4.15}$$

In general, each D_i involves several constants and, therefore, the number of positions where the measurements are taken must at least match the number of the constants to obtain enough number of independent linear simultaneous equations to solve for c_{ij}. Care must be taken also of how the joint being measured moves. The relationships in Table 4.1 are valid only when there is a non-zero velocity; any current readings taken while the joint is stationary correspond to region between the two curves which is highly nondeterministic. Configurations of PUMA manipulator used to measure the gravitational constants are listed in Table 4.2. Entries left blank are not important for the measurements.

Table 4.2 Configurations for c_{ij} Measurement

Configurations						*Resulting Equation*
θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	
	0	0				$\mathrm{D}_3 = c^l_{31}$
	-90	90				$\mathrm{D}_3 = c^l_{31}$
	0	90				$\mathrm{D}_3 = c^l_{32}$
	-90	180				$\mathrm{D}_3 = c^l_{32}$
	0	180				$\mathrm{D}_2 = c^l_{21} - c^l_{31}$
	180	180				$\mathrm{D}_2 = c^l_{21} + c^l_{31}$
	-90	180	90	90		$\mathrm{D}_4 = -c^l_{50}$
	-90	0	90	90		$\mathrm{D}_4 = -c^l_{50}$

As the result, the gravitational constants for the PUMA 560 in the experiment have been determined to be:

$$\begin{aligned} c^l_{21} &= 6373.(oz-in) \\ c^l_{31} &= -80.(oz-in) \\ c^l_{32} &= 1130.(oz-in) \\ c^l_{50} &= -70.(oz-in) \end{aligned} \qquad (4.16)$$

The gravity loading terms D_i can be evaluated easily with these constants from dynamics compensation. Experiments have proved the results to be satisfactory.

5.3. Effective Inertias

The knowledge of the effective inertias is important in dynamics compensation when the joints experience acceleration or deceleration. The approach to their determination is similar to that used for measuring gravitational constants except, in this case, the situation is complicated by the fact that the instantaneous joint accelerations are not directly obtainable. Dynamics Eq. (4.2) when all joint accelerations and velocities are zero except for joint i has the form:

$$\tau_i = (Ia_i + D_{ii})\ddot{\theta}_i + D_i \qquad (4.17)$$

The gravity loading term D_i is available from the previous section and if it is moved to the right-hand side, the equation becomes

$$\tau_i - D_i = (Ia_i + D_{ii})\ddot{\theta}_i \qquad (4.18)$$

It is noted in the above equation that since D_{ii} is always a function of only θ_j with $j>i$, if all joints except for joint i remain stationary, the coefficient of the acceleration on the left-hand side of the above equation is not time-varying. To make use of this property, Eq. (4.14) can be integrated twice with respect to time. Physically, Equation (4.14) represents Newton's Law as applied to a manipulator joint. The first integration produces linear or angular momentum and the second intergration produces work done by the joint. The evaluation of the integration of the right-hand side then becomes computation of work, requiring only the link inertia multiplied by the traveled distance or the difference between the initial and terminal joint positions, which are easily obtainable. The evaluation of the double integration of the left-hand side can be performed numerically as a double summation with the instantaneous values of the intergrand known at all times.

Let ΔT stand for the sampling period and the integration be performed over n sampling periods. The left-hand side after the first integration becomes a momentum M and

$$M(m\Delta T) = \int_0^{m\Delta T} (\tau_i - \mathrm{D}_i)dt \approx \sum_{k=0}^{m} (\tau_i(k) - \mathrm{D}_i(k))\Delta T \tag{4.19}$$

To integrate the above equation again, work $W(t)$ is obtained

$$W(n\Delta T) = \int_0^{n\Delta T} M(t)dt \approx \sum_{m=0}^{n} M(m\Delta T)\Delta T \approx \sum_{m=0}^{n} \left[\sum_{k=0}^{m} (\tau_i(k) - \mathrm{D}_i(k))\Delta T \right] \Delta T \tag{4.20}$$

Integrating the right-hand side of Eq. (4.14) and assuming that both Ia_i and D_{ii} are not functions of time, one first gets

$$\int_0^t (Ia_i + \mathrm{D}_{ii})\ddot{\theta}_i dt = \int_0^t (Ia_i + \mathrm{D}_{ii})d\dot{\theta}$$

$$= (Ia_i + \mathrm{D}_{ii})(\dot{\theta}_i(t) - \dot{\theta}_i(0)) \tag{4.21}$$

Assume that $\dot{\theta}_i(0)=0$ and integrate the above equation over time again, one gets

$$\int_0^{n\Delta T} (Ia_i + \mathrm{D}_{ii})\dot{\theta}_i(t)dt = (Ia_i + \mathrm{D}_{ii})(\theta_i(n\Delta T) - \theta_i(0)) \tag{4.22}$$

Combining equations (4.16), (4.18), and (4.20) leads to

$$(Ia_i + \mathrm{D}_{ii})(\theta_i(n\Delta T) - \theta_i(0)) \approx \sum_{m=0}^{n} \left[\sum_{k=0}^{m} (\tau_i(k) - \mathrm{D}_i(k))\Delta T \right] \Delta T$$

or

$$(Ia_i + \mathrm{D}_{ii}) \approx \frac{\sum_{m=0}^{n}\left[\sum_{k=0}^{m}(\tau_i(k) - \mathrm{D}_i(k))\Delta T\right]\Delta T}{(\theta_i(n\Delta T) - \theta_i(0))} \tag{4.23}$$

provided that $\theta_i(n\Delta T) \neq \theta_i(0)$.

In Eq. (4.19), the left-hand side is an expression in terms of dynamics constants to be determined, if positions all joints outer to link i where D_{ii} is being measured remain unchanged. Of the terms on the right-hand side, θ_i and $\theta_i(n\Delta T)$ are the initial and final joint positions, instantaneous $\tau_i(k)$ can be calculated by the relationship between joint torque and current, and $\mathrm{D}_i(k)$ can be evaluated using the gravitational constants measured above from the arm configuration at each time instant. Equation (4.19), therefore, is the basic relation followed to calculate the inertial constants b_{ij}.

The details of measurement made on PUMA 560 manipulator are described by the following sets of equations, with set i corresponding to the measurement of D_{ii}. The joints positions not specified in a set of equation are irrelevant to that measurement. Again, enough number of independent equations must be made available for solving constants of each link. In these equations, each $b^l_{i0}+Ia_i$ is treated as one constant, because they are inseparable.

Joints 5 and 6 are simple and one equation is sufficient for each link. For joint 6,

$$\mathrm{D}_{66} = b^l_{60} + Ia_6 \tag{4.24}$$

and for Joint 5

$$\mathrm{D}_{55} = b^l_{50} + Ia_5 \tag{4.25}$$

When determining constants in D_{44}, θ_j where $j > 4$ must not move.

$$\mathrm{D}_{44} = \begin{cases} b^l_{40} + Ia_4 & \theta_5 = 0 \\ b^l_{40} + Ia_4 + b^l_{41} & \theta_5 = 90 \end{cases} \tag{4.26}$$

When determining constants in D_{33}, θ_j where $j>3$ must not move.

$$D_{33} = \begin{cases} b^l_{30}+Ia_3 & \theta_5 = 90 \\ b^l_{30}+Ia_3+b^l_{31} & \theta_5 = 0 \end{cases} \tag{4.27}$$

When measuring constants in D_{22}, θ_j where $j>2$ must not move.

$$D_{22} = \begin{cases} b^l_{30}+Ia_2+b^l_{26} & \theta_3 = 90 \\ b^l_{20}+Ia_2+b^l_{25} & \theta_3 = 0 \\ b^l_{20}+Ia_2-b^l_{25} & \theta_3 = 180 \end{cases} \tag{4.28}$$

D_{11} is the most complicated and, when the constants in it are measured, θ_j where $j>1$ must not move for each chosen configuration and, since there are six constants to be determined, a total of six independent equations are necessary.

$$D_{11} = \begin{cases} b^l_{10}+Ia_1 & \theta_2 = -90,\ \theta_3 = 0 \\ b^l_{10}+Ia_1+b^l_{11}+b^l_{12}+b^l_{13} & \theta_2 = 0,\ \theta_3 = 0 \\ b^l_{10}+Ia_1+b^l_{12} & \theta_2 = 90,\ \theta_3 = 90 \\ b^l_{10}+Ia_1+b^l_{11}+b^l_{14} & \theta_2 = 0,\ \theta_3 = 90 \\ b^l_{10}+Ia_1+\tfrac{1}{2}b^l_{12}+\tfrac{1}{2}b^l_{15} & \theta_2 = 45,\ \theta_3 = 45 \\ b^l_{10}+Ia_1+\tfrac{1}{2}b^l_{11}+\tfrac{1}{2}b^l_{12}-\tfrac{1}{2}b^l_{13}+\tfrac{1}{2}b^l_{14}-\tfrac{1}{2}b^l_{15} & \theta_2 = 90,\ \theta_3 = 90 \end{cases} \tag{4.29}$$

The inertial constants for PUMA 560 measured by the above equations have been determined as

$$\begin{aligned} & b^l_{10}+Ia_1 = 522,\ b^l_{11} = 121,\ b^l_{12} = 26,\ b^l_{13} = 86,\ b^l_{14} = 141,\ b^l_{15} = 22 \\ & b^l_{20}+Ia_2 = 1757,\ b^l_{25} = 23,\ b^l_{26} = 131 \\ & b^l_{30}+Ia_3 = 317,\ b^l_{31} = 79 \\ & b^l_{40}+Ia_4 = 43,\ b^l_{41} = 5 \\ & b^l_{50}+Ia_5 = 31 \\ & b^l_{60}+Ia_6 = 67 \end{aligned} \tag{4.30}$$

The coupling inertias, unfortunately, can not be as easily determined as Eq. (4.22), for D_{ij} is a function of θ_j and, to measure constants in D_{ij}, θ_j must be in motion. Consequently, when Eq. (4.17) with non-zero $\ddot{\theta}_j$ is integrated, constants in D_{ij} can no longer be isolated. If integration were to be performed numerically, all constants would be considered at the same time and one would have to be able to read instantaneous joint acceleration.

6. Conclusion

This Chapter discusses the dynamics problem in manipulator control. Among different methods available for computing dynamics, Lagrangian equations proves to be the most appropriate for manipulator control. Due to the nature of the equations. the formulation can be easily implemented on a parallel computer with interactions among processors minimized. Its closed form makes it possible to derive the equations symbolically and compute them in background at a rate slower than the joint servo rate, thereby reducing real-time computational complexity considerably. The symbolic equations can also be simplified based on significant analysis; in particular, the velocity dependent terms D_{ijk} can be justly ignored. Using the method introduced in this Chapter, the constants in the dynamics equations can be obtained experimentally for applications. PUMA 560 manipulator is used as an example to demonstrate the developed methods.

In all the above discussions, any load carried by the manipulator is not considered in the dynamics calculation. Dynamics equations with load taken into account can be treated by first calculating the equations without a load and then adding to them a second set of equations due to load only. The detailed discussion can be found in [Izaguirre and Paul 1984].

CHAPTER V

COMPLIANCE AND HYBRID CONTROL

1. Introduction

The previous chapters dealt with control of robot manipulators in free space where it was assumed that manipulators are not required to make physical contact with the environment. However, this is usually not the case in many manipulator tasks. Assembly, for example, requires physical interaction between the manipulator and the parts to be assembled. Two problems naturally arise from these tasks: first, when contact is first made between the manipulator and a part, how is the contact achieved in a collision free manner to avoid damage that would otherwise result to both the manipulator and the part; secondly, once the contact is made, how does a manipulator handle geometric constraints presented by the task which prevent it from moving in certain directions while performing the task.

The importance for a robot manipulator to control force has long been recognized [Inoue 1971, Paul 1972, Paul and Shimano 1976, and Raibert and Craig 1981] [Groom 1972, Silver 1973, Whitney 1977, and Salisbury 1980]. As an alternative to pure position control, a manipulator actively monitors the contact forces to overcome its inability under the circumstances where it cannot possibly perform tasks constrained by environment by its limited positioning accuracy. By making use of the large forces developed between the manipulator and the parts, a manipulator can correct or modify positions in such a way as to minimize the contact forces presumably caused by the positional errors. In many cases, the tolerance of the task is usually of order of several tenths of a millimeter, while the contact forces due to position errors could easily reach several kilograms. Thus, treating geometric constraints as a guidance rather than obstacles enables a manipulator to succeed in these tasks.

Two distinct force control or compliance strategies, passive mechanical compliance and active programmed compliance, have been developed for a manipulator to be capable of reacting or complying to contact forces. The former strategy relies on special mechanical devices which comply to external forces in certain directions [Lord 1983]. The technique is limited by both its range and its dependency on applications. When a new task is to be performed or when the center of compliance changes, new compliance devices are necessary. On the other hand, the other more flexible strategy, the programmed compliance, is built into the manipulator control system. Information on the external forces collected by the sensing system is used to modify the motions. This way, it is possible to change the compliance required by the task geometry, as the task proceeds, as well as the center of compliance.

Among various methods proposed for manipulator active force control, they can be classified as explicit force feedback approaches and hybrid control approaches and there are various problems associated with each of the proposed approaches. A stiffness approach [Whitney 1985] computes reaction forces that a manipulator should apply in response to geometric constraints, but because of this, it will fail near singularity regions where manipulator is unable to apply Cartesian forces by its joints. The hybrid control method will not fail near the singularity regions, but since Jacobian inverse must be computed, the control strategy must be designed to be efficient in order to be applied for real-time control. This Chapter examines the representative force control schemes and proposes a modified hybrid control method. The stability of Cartesian force control systems, which has not yet been treated in any other literature, will also be investigated. Finally, the theoretical development is demonstrated by an implementation on PUMA 560 manipulator.

2. Compliance Specification

Mason summerized the work in the area of compliance specification [Mason 1979]. The contact of interest between the manipulator and the environment is that between the end effector and the part being manipulated. Given a manipulator task, a Cartesian coordinate frame can be chosen as *compliance frame* so that the six degrees of freedom in the frame are partitioned into two orthogonal complement sets, with directions in one set constrained by the task geometry and the directions in the other set unconstrained. This partition can be expressed by a six-by-six diagonal selection matrix **S**, whose ith diagonal element, boolean s_i, specifies the constraint of ith direction, as in Eq. (5.1).

$$\mathbf{S} = \begin{bmatrix} s_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & s_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & s_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & s_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & s_6 \end{bmatrix} \tag{5.1}$$

If ith direction is constrained, $s_i = 1$ and, if it is not, $s_i = 0$. Consider, for example, the case of peg-in-hole illustrated in Figure 1.1. The compliance frame is depicted with the z axis aligned with the hole axis. The motion is allowed only along and about z axis, but neither along nor about both x and y directions. Therefore, the corresponding selection matrix has the form

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

Associated with a selection matrix is its complement which has ones in directions along or about which force is constrained, i.e., no force can be applied in those directions because of the absence of the geometric constraints. To specify a task, two sets of trajectories must be specified. In constrained directions, a set of force trajectories define how forces will be applied; in unconstrained directions, motion trajectories are specified in much the same way as motion specification in free space except a few degrees of motion freedom have been lost. Two extreme cases occur, when a manipulator is free to move in any directions but cannot apply any forces, and when a manipulator tip is set in concrete so that no motion is possible in any directions but arbitrary forces can be exerted.

The choice of center of compliance or compliance frame is particularly important in a task specification. The compliance frame may have to change from motion to motion, as does the compliance requirement, and it must be so chosen that the degrees of freedom can be readily partitioned into a motion degree of freedom set and a force degree of freedom set. Tracing a corner in Figure 5.1, for example, requires the manipulator to comply in y and about z and x directions and control position in the rest of directions when the block travels along AB side until the end of AB side is reached, at which time the block starts to trace BC side, compliance directions are switched to along x and about z and y, and the other directions become position controlled.

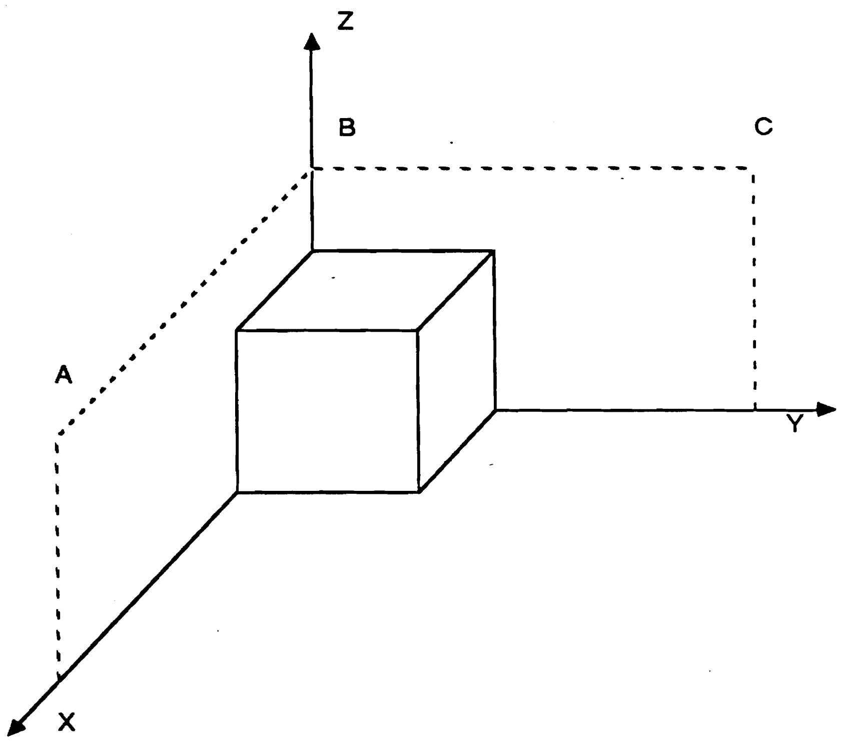


Figure 5.1. Tracing a Corner

3. Representatives of Active Force Control

A number of methods have been proposed and implemented for manipulator force control applications through manipulator's active reaction to contact forces, as opposed to passive response of mechanical devices, to perform tasks with compliance requirement. The fashion in which a manipulator reacts to the external contact forces can also be different. There are mainly two distinct approaches, namely the explicit force feedback or stiffness control approach [Groom 1972, Silver 1973, Whitney 1977, and Salisbury 1980] and hybrid control approach [Inoue 1971, Paul and Shimano 1976, and Raibert and Craig 1981].

To understand different approaches, consider first a simplified manipulator system that controls pure position in Figure 5.2.

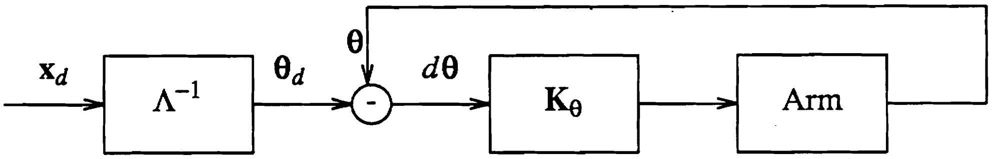


Figure 5.2. A Pure Position Controller

A Cartesian set-point $\mathbf{X}_d$ is converted to joint set-point $\boldsymbol{\theta}_d$ by the inverse kinematics Λ^{-1}. Errors $d\boldsymbol{\theta}$ is obtained by comparing θ_d and the actual joint position $\boldsymbol{\theta}$. $d\boldsymbol{\theta}$ is then fed into the joint servo controller, which calculates corrective torques to be applied to joint. For the sake of simplicity, a joint servo contains only a stiffness term and other possible terms such as velocity damping and integral term are ignored. A manipulator control system consists of n such controllers, working independently to control individual joints.

In the explicit force feedback approach, compliance strategy is built upon the thought that passive compliance of the mechanical systems can be achieved by the manipulator if it can react differently in terms of force in different directions. In directions where compliance is needed, weak Cartesian force reactions or stiffnesses are maintained by the controller, thereby providing compliance in these directions. In directions where no geometric constraints exist, the manipulator maintains high stiffnesses. Conceptually, this makes a manipulator behave like a six-dimensional spring with different stiffnesses defined for six Cartesian directions. In one of the most successful and complete reports based on the above approach, Salisbury [1980] implemented the idea and demonstrated how a joint actuated manipulator can accomplish the Cartesian stiffness specifications. Its simplified version with no stability consideration can be illustrated by the block diagram in Figure 5.3.

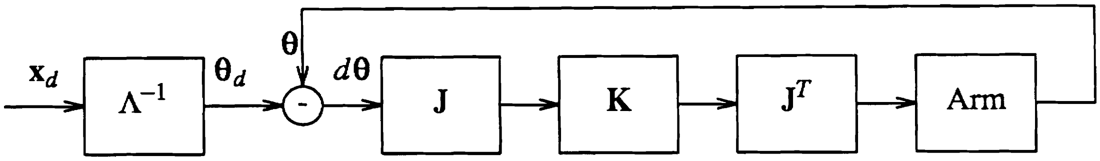


Figure 5.3. Stiffness Control Method

To achieve programmable compliance in Cartesian space, Cartesian stiffness characteristics of the manipulator are defined by a diagonal stiffness matrix **K**, one similar to the selection matrix **S** except the diagonal elements are no longer booleans but real numbers representing the desired stiffnesses in six directions. Cartesian reaction forces **f** then equal the Cartesian errors $d\mathbf{x}$ multiplied by **K**, i.e,

$$\mathbf{f} = \mathbf{K}\mathbf{x} \tag{5.3}$$

where $\mathbf{K} = diag[k_1, k_2, \cdots, k_6]$. The joint reaction torques are obtained by Eq. (2.3) as

$$\tau = \mathbf{J}^T\mathbf{f} \tag{2.14}$$

The joint reactions torques then drive the individual joints. To compute the Cartesian errors, joints errors are multiplied by **J**, i.e.,

$$d\mathbf{x} = \mathbf{J}d\theta \tag{5.4}$$

The final formulation becomes

$$\tau = \hat{\mathbf{K}}_\theta d\theta \tag{5.5}$$

where

$$\hat{\mathbf{K}}_\theta = \mathbf{J}^T\mathbf{K}\mathbf{J} \tag{5.6}$$

is defined by Salisbury as *joint stiffness matrix.*

This method deals with compliance directly in the Cartesian space where compliance is specified. Cartesian reaction forces, however, are realized in the joint space. The method is computational efficient since the joint stiffness matrix $\hat{\mathbf{K}}_\theta$, which is not diagonal in general, can be computed in background at a rate slower than the joint servos and it is possible to control the joints in integer arithmetics by properly scaling $\hat{\mathbf{K}}_\theta$.

In contrast, hybrid control approach treats the degrees of freedom of the Cartesian space as those strictly controlled in force and those strictly controlled in position. At joint level, manipulator is perfectly stiff to control positions. An early implementation of this approach was attempted by [Paul and Shimano 1977]. In Paul and Shimano's free-joint method, the compliance specification $\mathbf{S}$ given with respect to Cartesian compliance frame is matched as closely as possible to a joint compliance selection matrix $\mathbf{S}_\theta$, which is also diagonal with 1s and 0s and has a dimension equal to the number of joints. Joint errors $d\theta$ are modified by multiplying with $\bar{\mathbf{S}}_\theta$ first before multiplied by the diagonal joint stiffness matrix $\mathbf{K}_\theta$, as illustrated in Figure 5.4.

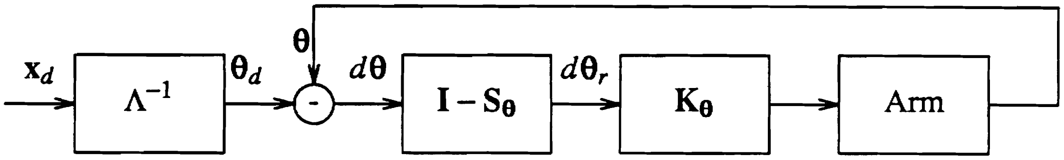


Figure 5.4. Free-Joint Method

Given a compliance requirement, a matching process determines one joint for each compliant direction specified, on the basis of joints' ability to most closely provide the compliance. Joint space then is partitioned into force servoed subset and position servoed subset. In general, an exact partition implies that the set of *n* joints is divided into two subsets, each of which contains a sufficient number of joints to provide positional degrees of freedom or compliant degrees of freedom, respectively, of the Cartesian space. Only when the matching is perfect does the method give an exact solution. This would be the case, for example for the PUMA 560, if the manipulation is to comply in rotation in the approach direction which happens to be the *z* axis about which joint six rotates so that compliance in the direction is perfectly provided. When the partition is not perfect, position errors due to force servoed joints are computed and compensated for by the position servoed joints continuously.

The method is nonetheless simple and efficient with joints servoed independently. Although the method is at best an approximate one, it has been successful in many cases because the geometry of the task can usually well fit into the geometry of the manipulator joints, allowing almost perfect partition of the joint space, when some manipulator joints are aligned with the desired compliance directions.

Craig and Raibert proposed an improved hybrid control method [Raibert and Craig 1981] based on Paul and Shimano's and Mason's work. A simplified version of the approach is illustrated in Figure 5.5.

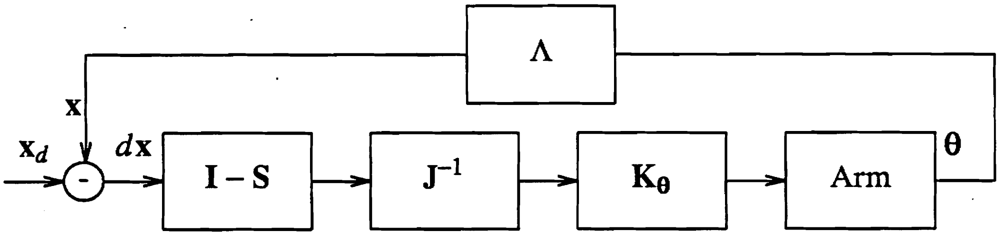


Figure 5.5. Hybrid Control Method

The approach makes use of the selection matrix **S** to compute Cartesian errors in position controlled directions

$$d\mathbf{x}_r = (\mathbf{I} - \mathbf{S})d\mathbf{x} \tag{5.7}$$

where $d\mathbf{x}$ is the difference between the actual Cartesian positions and the disired positions. Position errors in compliant directions are thus ignored. The real errors $d\mathbf{x}_r$ are then converted back to joint space for joint servos by inverse Jacobian matrix $\mathbf{J}^{-1}$. The formulation then becomes

$$d\theta_r = \mathbf{J}^{-1} d\mathbf{x}_r = \mathbf{J}^{-1}(\mathbf{I} - \mathbf{S})d\mathbf{x} \tag{5.8}$$

To calculate reaction torques, joint stiffness matrix is used as in Eq. (5.9).

$$\tau = \mathbf{K}_\theta \; d\theta_r \tag{5.9}$$

The method asserts the manipulator is still a positioner. Cartesian space and joint space interact by way of position errors not reaction forces. Further an infinite stiffness is also maintained at each manipulator joint.

4. Modified Hybrid Control Method [Zhang and Paul 1985]

The different methods discussed in the previous section reflect the general approaches to force control or compliance. Drawbacks, however, do exist in these methods. The free-joint method is approximate in nature and will not perform well in general situations. Its application is restricted to special tasks. Active stiffness control experiences problems when the manipulator *approaches* singularities. This can be demonstrated by the following example of a simple manipulator in Figure 5.6.

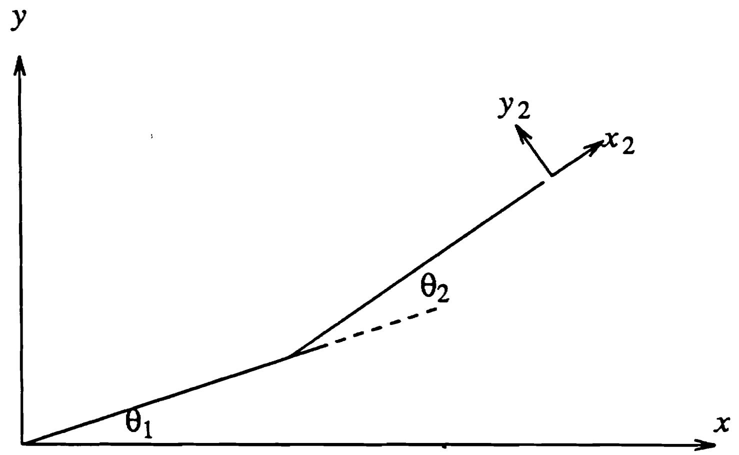


Figure 5.6. A Simple Manipulator

The manipulator has two links of unit length and two revolute joints. The manipulator has the differential relationship relating x_2 and y_2 to θ_1 and θ_2:

$$\begin{bmatrix} dx_2 \\ dy_2 \end{bmatrix} = \begin{bmatrix} S_2(1+C_2) & 0 \\ C_2(1+C_2) & 1 \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \end{bmatrix}$$

The manipulator has a singularity point when its θ_2 approaches zero, in which case, $sin(\theta_2) \approx \theta_2$, $\cos(\theta_2) \approx 1$, and the manipulator Jacobian matrix becomes

$$\mathbf{J} \approx \begin{bmatrix} 2\theta_2 & 0 \\ 2 & 1 \end{bmatrix}$$

and assume $\mathbf{K} = diag\,[k_x,\ k_y]$, the joint stiffness matrix equals

$$\hat{\mathbf{K}}_\theta = \mathbf{J}^T\,\mathbf{K}\,\mathbf{J} = \begin{bmatrix} 2\theta_2 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \begin{bmatrix} 2\theta_2 & 0 \\ 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4\theta_2^2 k_x & 2k_y \\ 2k_y & k_y \end{bmatrix}$$

If the manipulator is to control x_2 in position and y_2 in force, $k_y \approx 0$ and the joint stiffness matrix becomes

$$\hat{\mathbf{K}}_\theta = \begin{bmatrix} 4k_x\theta_2^2 & 0 \\ 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \end{bmatrix} \quad as\ \theta_2 \rightarrow 0$$

No restoring torques will be applied whatsoever because of the null joint stiffness matrix, even though position errors may occur in the position controlled x_2 direction.

The reason for this inability of manipulator to control position in regions *around* the singularities lies in that the formulation relies on reaction forces to control positions, but, unfortunately, at those regions the manipulator is not capable of applying Cartesian forces, even though position errors can still be corrected.

Craig and Raibert's hybrid control method, on the other hand, does not experience such regions. In this case, the Cartesian reaction errors rather than reaction forces are computed and then converted back to the joint space by the inverse Jacobian $\mathbf{J}^{-1}$. The joint errors are then multiplied by the joint stiffness to obtain joint reaction torques. For the simple manipulator in Figure 5.6, the selection matrix for compliance in y_2 and position control in x_2 is

$$\mathbf{S} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

and the joint reaction torques are

$$\tau = \mathbf{K}_\theta \, \mathbf{J}^{-1}(\mathbf{I} - \mathbf{S})d\mathbf{x}$$

where, for the purpose of comparison with stiffness control, let $d\mathbf{x} = \mathbf{J}d\boldsymbol{\theta}$. The inverse Jacobian of our simple manipulator equals

$$\mathbf{J}^{-1} = \begin{bmatrix} \dfrac{1}{S_2(1+C_2)} & 0 \\ -\dfrac{C_2}{S_2} & 1 \end{bmatrix}$$

and, for small θ_2 it becomes

$$\mathbf{J}^{-1} = \begin{bmatrix} \dfrac{1}{2\theta_2} & 0 \\ -\dfrac{1}{\theta_2} & 1 \end{bmatrix}$$

and the joint reaction torques are

$$\tau = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix} \begin{bmatrix} \dfrac{1}{2\theta_2} & 0 \\ -\dfrac{1}{\theta_2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2\theta_2 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \end{bmatrix}$$

$$= \begin{bmatrix} k_1 d\theta_1 \\ -2k_2 d\theta_1 \end{bmatrix}$$

where k_i is the stiffness of joint i. The restoring torques will not approach zero even when the manipulator approaches singularity. Error in position can only be caused by error in θ_1, which will be corrected. Error in θ_2 will not cause error in x_2 and, therefore, will not be corrected. The effectiveness of the method is due to the fact that position errors maintain a better integrity in regions of singularity than the forces, and they are used in Cartesian space as a measure of how joints should react, as pointed out earlier.

The hybrid control method is correct theoretically. In practice, however, problem exists in the proposed controller architecture. Calculation of errors is performed in Cartesian space and, therefore, has to be carried out in real numbers, and so does the next matrix/vector multiplication converting Cartesian errors back to joint space. Further, the inclusion of the costly inverse kinematics in the feedback loop causes it evaluated every sampling period. All of the computation presents a computational load inappropriate for real-time control. To overcome the computational problem, the controller illustrated in Figure 5.7 is proposed.

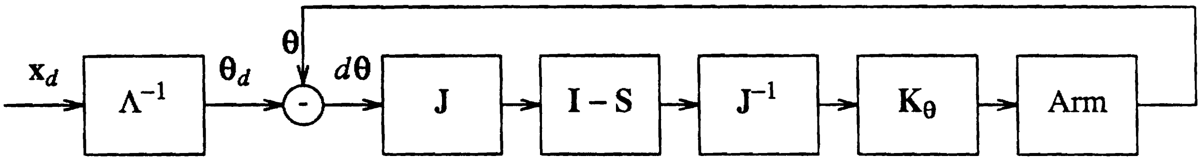


Figure 5.7. Modified Hybrid Control Method

Cartesian errors are calculated by transforming joint errors by the Jacobian matrix instead of comparing desired and actual Cartesian positions. Reaction Cartesian errors are obtained by multiplying $d\mathbf{x}$ by I–S. They are converted back to joint space by the Jacobian inverse matrix to be multiplied by the joint stiffnesses. The final joint reaction torques become

$$\tau = \mathbf{K}_\theta \mathbf{J}^{-1}(\mathbf{I}-\mathbf{S})\mathbf{J}d\theta \tag{5.10a}$$

The inverse kinematics is eliminated from the feedback loop in this case and, similar to the active stiffness control of Figure 5.3, the matrix product,

$$C_\theta = J^{-1}(I-S)J \tag{5.10b}$$

defined as *joint compliance matrix*, can be computed in background and properly scaled for integer arithmetics.

Viewed from individual joints, errors in joint space are decomposed by the joint compliance matrix $J^{-1}(I-S)J$ into those to correct positions errors in non-compliant directions of the Cartesian space and those to provide compliance in compliant directions.

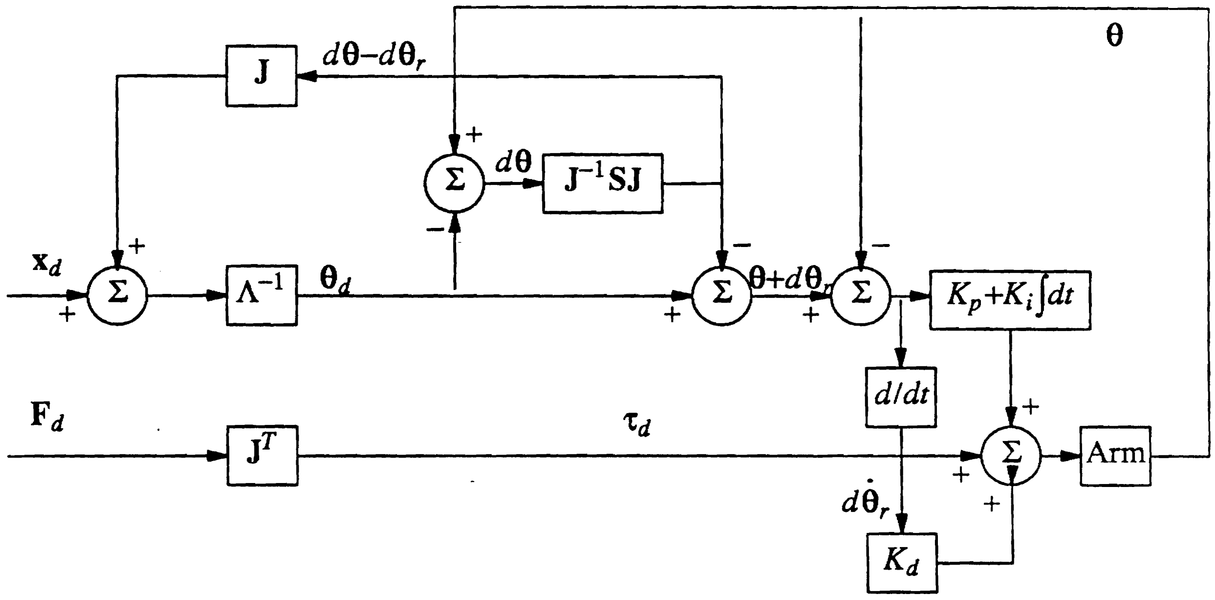
5. Implementation

The modified hybrid control method was implemented to control a PUMA 560 manipulator [Zhang and Paul 1985], as illustrated in Figure 5.8. The task specification is embedded in RCCL [Hayward and Paul 1984] to allow flexible programming of the force control applications. There are a number of issues to consider when implementing the modified hybrid controller.

The Cartesian position errors ignored by $\bar{S}$ is fedback to the trajectory generator in order to take it into consideration when producing the subsequent set-points. This is performed by computing a **COMPLY** transformation, derived by Eq. (2.16) from ignored Cartesian errors, and inserting it after the transformation corresponding to the compliance frame in the position equation,

$$\begin{aligned} &\mathbf{COMPLY}(t+\Delta t) = \mathbf{COMPLY}(t)\ \Delta(t) \\ &T_6 = \mathbf{L}\ \mathbf{D}(r)\ \mathbf{COMPLY}\ \mathbf{R} \end{aligned}$$

At times the manipulator needs to exert biased forces in some directions in addition to complying in those directions. This is necessary, for example, when the manipulator is to maintain contact with a surface. Biased force is applied in the direction perpendicular to the surface where contact is made so that stable contact is obtained. Given the desired contact force as a vector in the compliance frame, it is converted to



5.8. MHCM Controller Design

joint space by the Jacobian transpose $\mathbf{J}^T$ as in Eq. (2.20) and then added to the joint torque correcting joint position errors. While the position is controlled in a closed loop, the force control loop can be either closed or open. Performance of force application can be improved by using closed-loop control, but at the same time the scheme requires costly force sensing devices either at joints or at manipulator wrist and additional computations. In case force sensors are available, applied force is compared with the desired force as force error. Similar to positional error, joint force error is multiplied to a gain before added to the total joint torque.

In the implementation, the joint servo must provide stable and accurate position control. A single stiffness term or a proportional gain is not sufficient, and usually proportional, integral, plus derivative (PID) control or proportional plus derivative (PD) is employed for this purpose. Derivative term stabilizes the system and integral term eliminates steady state errors. Although there exist methods to determine the optimal combination of controller gains theoretically for each joint, the gains can be determined based on simpler criteria and on experiments as well.

Dynamics such as gravity loading and inertias has also to be compensated, as does the joint frictions. The knowledge of the manipulator dynamics and frictions has been obtained with the method introduced in Chapter IV. The final joint torque to be applied then consists of all the above contributions and, if open force control loop is used, has the form

$$\tau_i = K_{p_i} d\theta_{ri} + K_{i_i} \int d\theta_i dt + K_{d_i} d\dot{\theta}_{ri} + f_i + D_i + T_{f_i} \tag{5.11}$$

where

K_{p_i} = proportional gain of joint i
K_{i_i} = integral gain of joint i
K_{d_i} = derivative gain of joint i
f_i = static friction of joint i
D_i = gravity loading of joint i
T_{f_i} = joint torque for biased force

On the other hand, if closed force control loop is desired, the actual torque would be compared to the desired torque T_{f_i}. Torque error would then be multiplied by a gain to be added to τ_i.

As an example, the following RCCL program performs inserting a peg into a hole once the peg is in the hole.

```
#include "rccl.h"   /* RCCL system file */

pumatask()
{
     TRSF_PTR z, e, b1, b2;
     POS_PTR  top, bottom;

     z = gentr_trsl("Z",  0.,  0., 864.);
     e = gentr_trsl("E",  0.,  0., 200.);
```

```
    b1 = gentr_rot("B1", 600. , 250., 700., yunit, 180.);
    b2 = gentr_rot("B2", 600. , 250., 500., yunit, 180.);

    hole = makeposition("PT1", z, t6, e, EQ, b1, TL, e);
    bottom = makeposition("PT2", z, t6, e, EQ, b2, TL, e);

    move(hole);              /* get to hole */
    waitfor(completed);
    comply("fx fy rx ry",0., 0., 0., 0.);
    move(bottom);            /* with compliance spec above */
    lock("fx fz rx ry");     /* terminate the compliance    */
    release();           /* open the gripper */
    move(park);          /* go back home */
}
```

This is a much simplified version of an actual insertion program. Nonetheless, it shows how compliance can be specified and executed.

RCCL runs under Unix on a VAX 11/780, computing set points every 14 milliseconds or at 71 Hz. A real-time interface allows the VAX to exchange a 60-word buffer with the Unimation controller at each interrupt of the Unimation controller. At the end of every sampling period, the VAX sends the next set point in joint coordinates and the updated joint compliance compliance matrix before it receives the compliance compensations resulting from the control of the previous sampling period. The Puma controller controls the joints in parallel with the VAX and computes compliance compensations using sixteen-bit integer arithmetic.

The experiments show that the maximum force control errors occur when manipulator is at rest since joint frictions cannot be correctly compensated. On the average these errors are of the same order as the joint static frictions, bounded by one and a half pounds in translational directions and two and a half pound inches in rotational directions of the compliance frame. As the manipulator starts to comply in the specified direction, friction compensation term becomes effective and the manipulator enters a very smooth compliance mode.

6. Stability Consideration

From the control point of view, a robot manipulator is a multi-input and multi-output system. Each joint requires an input signal in terms of joint position or output force or torque and produces a position as its output. When input is in the form of position, joint servo system usually employs a simple PD, PI or PID control to generate corrective torque for the error in position. In applications where the manipulator needs only to control position, there is little interaction among joints. Therefore, joint controllers can be designed independently of each other and stability of individual joints assures the stability of whole manipulator system.

This, however, is not the case when manipulator needs to control both position and force. Interactions among joints occur through a non-diagonal matrix, i.e., input to a joint control no longer contains only one but n elements in general, where n is the number of joints of the manipulator. So far none of the methods proposed for manipulator force control have effectively dealt with this problem. It is always implicitly assumed that manipulator joint controllers are not affected by the proposed method.

Maples and Goldman [1984] noticed the problem in their study of Salisbury's stiffness control. However, no formal analysis was made. Motivated by the fact that if a manipulator joint is modeled as a double integrater

$$\tau = d_{ii}\ddot{\theta} \tag{5.12}$$

and if a PD control is used, the critical damping of the system can be set by $K_v = 2\sqrt{K_p\, d_{ii}}$, an ad doc scheme was suggested to continuously modify the velocity damping of joint i to

$$K_{v_i} = A\,\sqrt{K_{p_i}{}'} \tag{5.13}$$

with

$$K_{p_i}{}' = \sum_{j=1}^{n} |K_{p_{ij}}| \tag{5.14}$$

$K_{p_{ij}}$ are elements in the ith row of the joint stiffness matrix in Eq. (5.6), which represents new proportional gains that couple reacting torque at joint i with errors of all joints. Coefficient A is a universal fudge factor used to tune the system.

6.1. State Space Formulation

Systems employing the hybrid control method are faced by the same problem, as can be shown by Eqs. (5.10) where real joint error $d\theta_{ri}$ at one joint depends on errors of all joints. To study the stability of such systems in theory, one needs to establish a system model, on which the formal analysis is based. Assume that in a six-joint manipulator each joint has a simple PD control and the joint is modeled as a double integrater as in Eq. (5.12), a PD joint controller is defined by

$$\tau_i = K_{p_i} e_i + K_{d_i} \dot{e}_i \tag{5.15}$$

where the joint error $e_i = \theta_d - \theta$ and rate error $\dot{e}_i = \dot{\theta}_d - \dot{\theta}$. Further, define state variables

$$\mathbf{x} = [\mathbf{x}_1^T, \mathbf{x}_2^T]^T \tag{5.16}$$

where six-by-one vectors $\mathbf{x}_1 = \boldsymbol{\theta}$ and $\mathbf{x}_2 = \dot{\boldsymbol{\theta}}$. Define input vector as

$$\mathbf{U}^T = \begin{bmatrix} \theta_d \\ \dot{\theta}_d \end{bmatrix} \tag{5.17}$$

The state equations for a system employing pure position control are

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{U} \tag{5.18}$$

where **A** and **B** are six-by-six matrices and

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{D}^{-1}\mathbf{K}_p & -\mathbf{D}^{-1}\mathbf{K}_d \end{bmatrix}$$
$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ \mathbf{D}^{-1}\mathbf{K}_p & \mathbf{D}^{-1}\mathbf{K}_d \end{bmatrix} \qquad (5.19)$$

where

$$\mathbf{D}^{-1} = diag[1/d_{11},\ 1/d_{22},\ \cdots,\ 1/d_{66}]$$
$$\mathbf{K}_p = diag[K_{p_1},\ K_{p_2},\ \cdots,\ K_{p_6}] \qquad (5.20)$$
$$\mathbf{K}_d = diag[K_{d_1},\ K_{d_2},\ \cdots,\ K_{d_6}]$$

are the inverse of inertia matrix, proportional gain matrix, and derivative gain matrix, respectively. It is easy to verify that Eq(5.15) represents six decoupled joint PD controllers.

When compliance is required, however, both the position errors and velocity errors are modified by $\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J}$. The state equations for a hybrid controller have the new **A** and **B** matrices,

$$\dot{\mathbf{x}} = \mathbf{A}'\mathbf{x} + \mathbf{B}'\mathbf{U} \qquad (5.21)$$

where

$$\mathbf{A}' = \begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{D}^{-1}\mathbf{K}_p\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} & -\mathbf{D}^{-1}\mathbf{K}_d\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} \end{bmatrix}$$
$$\mathbf{B}' = \begin{bmatrix} 0 & 0 \\ \mathbf{D}^{-1}\mathbf{K}_p\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} & \mathbf{D}^{-1}\mathbf{K}_d\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} \end{bmatrix} \qquad (5.22)$$

To investigate the stability of the system, $\Delta(\lambda)$, the characteristic polynomial of the system, is derived first. This polynomial is given by $|\mathbf{A}' - \lambda\mathbf{I}|$ or

$$\Delta(\lambda) = \begin{vmatrix} -\lambda \mathbf{I} & \mathbf{I} \\ -\mathbf{D}^{-1}\mathbf{K}_p\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} & -\mathbf{D}^{-1}\mathbf{K}_d\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J}-\lambda\mathbf{I} \end{vmatrix}$$

$$= |\lambda\mathbf{D}^{-1}\mathbf{K}_d\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J} + \lambda^2\mathbf{I} + \mathbf{D}^{-1}\mathbf{K}_p\mathbf{J}^{-1}\bar{\mathbf{S}}\mathbf{J}| \tag{5.23}$$

Assuming all matrices are square and keeping in mind that modes of the system are roots of $\Delta(\lambda)$, Equation (5.23) can be simplified to

$$\Delta(\lambda) = |\mathbf{Q}(\lambda)\mathbf{J}^{-1}\bar{\mathbf{S}} + \lambda^2\mathbf{D}\mathbf{J}^{-1}\mathbf{S}| \tag{5.24}$$

where

$$\mathbf{Q} = diag[\lambda^2 d_{11}+\lambda k_{d_1}+k_{p_1},\ \lambda^2 d_{22}+\lambda k_{d_2}+k_{p_2},\ \cdots,\ \lambda^2 d_{66}+\lambda k_{d_6}+k_{p_6}]$$

$$= diag[q_1(\lambda), q_2(\lambda),\ \cdots, q_6(\lambda)] \tag{5.25}$$

$q_i(\lambda)$ is the joint characteristic function and, for a stable plant, it always has all eigenvalues with non-positive real parts.

The following observations can be made on $\Delta(\lambda)$: When this is no force control, $\mathbf{S} = [0]$ and $\bar{\mathbf{S}} = \mathbf{I}$, and the roots of the characteristic polynomial $\Delta(\lambda)$ are those of the the joint characteristic polynomials $q_i(\lambda)$. The stability of the joints then guarantees that of the whole system, as is expected in a pure position control when the dynamic interaction among joints is ignored. When all joints are force controlled, a so-called free situation, $\mathbf{S} = \mathbf{I}$ and $\bar{\mathbf{S}} = [0]$ so that

$$\Delta(\lambda) = |\lambda^2\mathbf{D}\mathbf{J}^{-1}\mathbf{S}| = \lambda^{12}\prod d_{ii}\,|\mathbf{J}^{-1}| \tag{5.26}$$

and all modes of the system have zero eigenvalues, implying the system is still stable in the sense of Lyapunov and the dynamics compensation becomes dominant.

When the situation is between the pure position and pure force control, the joints interact by way of the inverse Jacobian matrix. Consequently, the system stability depends on the manipulator kinematics and its current configuration. This makes it difficult to predict in general the outcomes of the controller as a result of configuration change. It can be concluded, however, that constant gains of joint controllers are not appropriate in this situation and that since the joints are usually designed with gain combinations to provide critical damping when working as decoupled systems, the critically damped joints will in general be impaired as the result of the coupling among joints to become either over-damped or under-damped or, even worst, unstable systems, if no adjustment is made to the gains.

Interestingly, hybrid controls have been implemented in a number of cases, and experiments have shown that the systems still remain stable. This seemingly contradictory result can be explained by the unique kinematics which a majority of today's robot manipulators possess.

6.2. Special Case

Most of the robot manipulators are made up of a three-link arm and a three-link wrist. The arm consists of a waist, a shoulder, and an elbow, and the wrist of three perpendicular intersecting axes with zero offsets. The arm provides translational degrees of freedom and the wrist provides rotational degrees of freedom in the Cartesian space. Further, an arm joint is coupled kinematically more with other arm joints than with wrist joints and a wrist joint is coupled only with other wrist joints. While the arm joints can change the manipulator orientation to certain extent, the wrist joints cannot change the manipulator position if no tool is attached to the end of manipulator. Such designs have the effect on the manipulator Jacobian matrix that if J is partitioned into four three-by-three blocks

$$J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{5.27}$$

then J_{11} relating the changes in arm joints to changes in manipulator position is highly non-diagonal, as is J_{22} relating the changes in wrist joints to those in manipulator orientation. For the off-diagonal blocks, J_{21}, relating the changes of arm joints to changes in manipulator orientation, has elements of small magnitude compared to J_{11}, while J_{12} is always null for manipulators with no tool.

The inverse Jacobian for such manipulators has the form

$$J^{-1} = \begin{bmatrix} J_{11}^{-1} & 0 \\ -J_{22}^{-1} J_{21} J_{11}^{-1} & J_{22}^{-1} \end{bmatrix} \tag{5.28}$$

In addition, since the *z* axis of the Cartesian is always the axis of θ_6, rotation change in that direction is solely provided by θ_6, causing all but the last element of column six of J^{-1} to be zero. With the special form of J^{-1}, Equation (5.22) can be further evaluated as the product of two determinants. Let

$$Q(\lambda) = \begin{bmatrix} Q_1(\lambda) & 0 \\ 0 & Q_2(\lambda) \end{bmatrix}$$

$$S = \begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix}$$

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

where every block matrix is three by three and diagonal, and let

$$H = -J_{22}^{-1} J_{21} J_{11}^{-1} \tag{5.29}$$

the characteristic polynomial

$$\Delta(\lambda) = \left| \begin{bmatrix} \mathbf{Q}_1(\lambda) & 0 \\ 0 & \mathbf{Q}_2(\lambda) \end{bmatrix} \begin{bmatrix} \mathbf{J}_{11}^{-1} & \mathbf{0} \\ \mathbf{H} & \mathbf{J}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{S}}_1 & 0 \\ 0 & \overline{\mathbf{S}}_2 \end{bmatrix} + \lambda^2 \begin{bmatrix} \mathbf{D}_1 & 0 \\ 0 & \mathbf{D}_2 \end{bmatrix} \begin{bmatrix} \mathbf{J}_{11}^{-1} & \mathbf{0} \\ \mathbf{H} & \mathbf{J}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{S}_1 & 0 \\ 0 & \mathbf{S}_2 \end{bmatrix} \right|$$

$$= | \mathbf{Q}_1(\lambda)\mathbf{J}_{11}^{-1}\overline{\mathbf{S}}_1 + \lambda^2 \mathbf{D}_1 \mathbf{J}_{11}^{-1}\mathbf{S}_1 | \, | \mathbf{Q}_2(\lambda)\mathbf{J}_{22}^{-1}\overline{\mathbf{S}}_2 + \lambda^2 \mathbf{D}_2 \mathbf{J}_{22}^{-1}\mathbf{S}_2 |$$

$$= \Delta_1(\lambda)\Delta_2(\lambda) \tag{5.30}$$

This shows that interaction among modes of joint controllers are confined to two sets – the arm joints can possibly be affected only by arm joints and the same holds for wrist joints. To show that stability of the system is totally independent of the manipulator configuration and of compliance specification, one needs an extremely strong assumption that the arm joints and the wrist joints have the similar characteristics, respectively, i.e.

$$\begin{aligned} d_{11} &\approx d_{22} \approx d_{33} \\ d_{44} &\approx d_{55} \approx d_{66} \end{aligned} \tag{5.31}$$

Under this assumption, PD controls for the arm joints and for the wrist joints are approximately equal, respectively. To evaluate $\Delta_1(\lambda)$ given the translational compliance, $\mathbf{S}_1 = diag[s_1, s_2, s_3]$, $\mathbf{S}_1$ is multiplied to $\mathbf{J}_{11}^{-1}$ on the right. The ith column of the resulting matrix is the ith column of $\mathbf{J}_{11}^{-1}$ if s_i is 1 and the ith column is a zero vector if s_i is 0. Similar matrix product results when $\mathbf{J}_{11}^{-1}$ is multiplied by $\overline{\mathbf{S}}_1$. If the two matrix products are multiplied on the left by diagonal matrices $\mathbf{Q}_1$ and $\lambda^2\mathbf{D}_1$, respectively. The final matrix has, as its ith column,

$$\begin{bmatrix} a_{1i}q_1(\lambda) \\ a_{2i}q_2(\lambda) \\ a_{3i}q_3(\lambda) \end{bmatrix} \approx \begin{bmatrix} a_{1i}q_1(\lambda) \\ a_{2i}q_1(\lambda) \\ a_{3i}q_1(\lambda) \end{bmatrix} \tag{5.32}$$

if $s_i = 0$. If $s_i = 1$, the ith column equals

$$\begin{bmatrix} a_{1i}\lambda^2 d_{11} \\ a_{2i}\lambda^2 d_{22} \\ a_{3i}\lambda^2 d_{33} \end{bmatrix} = \begin{bmatrix} a_{1i}\lambda^2 d_{11} \\ a_{2i}\lambda^2 d_{11} \\ a_{3i}\lambda^2 d_{11} \end{bmatrix} \tag{5.32}$$

where $\mathbf{J}_{11}^{-1} = [a_{ij}]$ and $1 \le i,j \le 3$. The $\Delta_1(\lambda)$ is then easy to calculate and has the form

$$\begin{aligned} \Delta_1(\lambda) &= |\mathbf{J}_{11}^{-1}|\,|q_1(\lambda)\bar{\mathrm{S}}_1 + \lambda^2 \mathrm{S}_1| \\ &= |\mathbf{J}_{11}^{-1}|\,q_1^{3-n}(\lambda)(d_{11}\lambda^2)^n \end{aligned} \tag{5.33}$$

where n is the number of ones on the diagonal of S_1. Therefore, all roots of $\Delta(\lambda)$ are then either 0, which gives the system a marginal stable mode, or roots of the q_1, which have negative real parts as assumed. Similar argument holds for $\Delta_2(\lambda)$ and it has the form

$$\begin{aligned} \Delta_2(\lambda) &= |\mathbf{J}_{22}^{-1}|\,|q_4(\lambda)\bar{\mathrm{S}}_2 + \lambda^2 \mathrm{S}_2| \\ &= |\mathbf{J}_{22}^{-1}|\,q_4^{3-m}(\lambda)(d_{44}\lambda^2)^m \end{aligned} \tag{5.34}$$

where m is the number of ones on the diagonal of S_2. The $\Delta(\lambda)$ becomes

$$\Delta(\lambda) = |\mathbf{J}^{-1}|\,q_1^{3-n} q_4^{3-m} \lambda^{2*(m+n)} d_{11}^n d_{44}^m \tag{5.35}$$

Although the assumption (5.31) is strong in general, many manipulators, such as PUMA 560, do have such features. Therefore, the assumption is not totally unreasonable. Secondly, the assumption is not necessary for some manipulator configurations and compliance specifications. There are cases where all modes of $\Delta(\lambda)$ are stable even without the assumption (5.31).

When there is a tool attached to the end of a manipulator, the Jacobian matrix does not have a null $\mathbf{J}_{12}$, the arm joints and the wrist joints of the manipulator become heavily coupled, and the stability of the system is highly unpredictable. This occurs to all joint-based control systems and has been demonstrated in a hybrid control method

[Zhang and Paul 1985] and in an adaptive force control method [Backes 1984].

6.3. Discussion

If a manipulator is to implement joint-based force control and if constant gains are employed, the resulting system is in general unstable due to the interaction among the joints. However, for manipulator with spherical wrist, if the center of compliance is chosen at the origin of the wrist axes, the system can be stable. Most of the manipulators are designed to have a spherical wrist, but there are many applications requring remote center of compliance, in which case, the system is bound to fail.

There is no known method yet to solve the problem properly. However, two approaches can be considered. First, the control system can vary joint gains in such a way that the system always have stable modes. This inevitably would require a lot of computations for determining how the gains should be adjusted. In the other approach, the compliance strategy is such that center of compliance is located at the the origin of the wrist no matter what task is being performed. This guarantees the system stability. The errors due to the change of the center of compliance, however, must be compensated for. Cartesian-based force control method [Burdick and Khatib 1984] could possibly offer yet another solution. Because of disparate nature of the formulation, it is difficult to make comparisons between Cartesian-based and joint based methods here.

7. Conclusion

The modified hybrid control method partitions the joint error vector into two components, one as real joint position errors and the other as compliant joint errors, according to a Cartesian compliance specification. This partition is accomplished by using the joint compliance matrix $\mathbf{C}_\theta$. The method is not only theoretically correct to overcome the inability of explicit feedback control methods in singularity regions, but also efficient computationally, for the joint compliance matrix can be scaled for integer arithmetics and computed in background, as can the dynamics equations of a manipulator, at a slower rate than the joint servos.

This theoretical development is demonstrated by the implementation on a PUMA 560 manipulator. The results show that the control accuracy in the experiment is limited only by the manipulator joint frictions. The experiments have also revealed the stability problem associated with the joint-based force controller employing constant gains. Preliminary investigation has proved that manipulators with similar arm joints and a spherical wrist can be stable if the center of compliance is at the center of the wrist. This opens up a new research interest in developing compliance strategies based on a fixed center of compliance at the wrist, and in designing controllers with time-varying gains.

CHAPTER VI

INTEGRATION OF THE RFMS

1. Introduction

A robot system consists of a number of components, each operating in its own domain and responsible for providing other system components with its services. A traditional industrial robot consists of a manipulator and a limited number of position sensors, such as joint encoders and tachometers, to provide information about joint positions and velocities for position control. As the requirement on the tasks a robot is to perform increases, so do the number and sophistication of the sensors. Sensors may appear in different forms, position, touch, force, vision, etc.; each sensor functions in its own domain of expertise and their observations are described geometrically [Paul, Durrant-Whyte and Mintz 1986]. A manipulator serves as a force and motion server (RFMS) to the robot. The robot, interpreting sensor information in terms of a world model and a task plan, issues instructions to the manipulator to carry out tasks [Brady 1984].

Computers used today in control of robot manipulators fall into two broad categories in terms of their architecture, those based on a supervisory process and a number of dedicated and decoupled joint processors, as is the case in many commercial industrial robots, and those based on a mini-computer, as is the case in a research or university laboratory environment where emphasis is often on the investigation of certain control algorithms for specific applications. Typical of the controllers in the former case, the PUMA controller [Unimation Inc. 1982] performs most of the computations on the supervisor and performs interpolation of set-points generated by the supervisor on the joint processors. Joint processors run at a high rate (1000 Hz) for smooth motions and the supervisor runs at a much lower rate (36 Hz) with a sampling period long enough to perform all necessary computations. Tasks performed by the manipulator are those of positional control without any sensor influence. Due to the lack of sophistication of the controller, only primitive applications such as pick-and-place operations are possible and the rigid design of these systems makes it difficult to modify the system to suit new control algorithms in general, and to control sensor driven motions in particular because of the delay caused by the slow Cartesian position update rate.

The systems based on a mini-computer [Hayward and Paul 1984] are usually designed not for general purpose manipulator control, but for some specific applications. The fact that the cost of the control computer cannot be economically justified limits the potential of such systems for wide practical applications in industry.

In recent years, as the microprocessor technology has quickly progressed, there is a growing interest in designing manipulator controllers based on multiple processors [Kriegman, Siegel, and Gerpheide 1985] [Nigam and Lee 1985] [Taylor, Korein, Maier, and Durfee 1985] [Turner, Craig, and Gruver 1986] [Paul and Zhang 1986]. Unlike the early industrial robot controllers, these systems employ powerful microprocessors and, more importantly, begin to address issues never dealt with before, including, for example, the design of parallel algorithms, resource sharing by processes, and interprocess communication required by Cartesian level control such as stiffness control and hybrid control, sensor integration, etc. Highly powerful systems are thus achieved, with desired system throughput at a reasonable cost.

In some proposed designs, the architecture of the controller is based on the dynamics equations of the manipulator [Nigam and Lee 1985] [Kasahara and Narita 1985]. The scheduling process of such systems is complex and, once finished, the systems are vulnerable to further modifications, when needed to suit new applications. In addition, systems based on dynamics equation alone cannot provide satisfactory performance in terms of positional accuracy; they must be incorporated with another feedback position controller, in which case a multiprocessor sub-system dedicated to dynamics computation is no longer a cost-effective solution.

Sensors play a critical role for robot to plan fine motions. The issue of sensor integration, however, is not always addressed in some controller designs [Kriegman, Siegel, and Gerpheide 1985, and Turner, Graig, and Gruver 1986]. They propose pipelined algorithms, causing long time delays and leading to systems that cannot control motions to be modified by sensors in real time.

This Chapter is concerned with construction of a controller for a robot manipulator to execute sensor driven tasks, according to decision made by the robot system as to what tasks to carry out next; however, it is not concerned with the coordination of the robot system nor with the integration of sensor observations, which are an interesting and challenging open research subject in its own right [Giralt 1984, Orlando 1984, and Paul, Durrant-Whyte, and Mintz 1986]. Information about the geometry of the objects in the world model collected by the sensors is simply assumed to be available to the manipulator control system. To achieve the goal, the resulting RFMS must maintain a high Cartesian update rate and be able to interface to sensor systems efficiently.

This Chapter first examines the basic requirements on the RFMS and identifies them in terms of processes; these processes are classified based on their real-time constraints. To minimize sensor feedback delay in the server and provide the robot with force/motion control abilities, a multi-processor computer architecture is proposed to distribute computations with processes assigned to processors. Interactions of the server with the robot coordinator and with sensors are studied. Finally, the system implementation using off-the-shelf single board computers is discussed and the system is constructed to provide a flexible force/motion server facilitating sensor controlled motions. To demonstrate the system, a trajectory generator and a hybrid controller are implemented to control a PUMA 250 manipulator. Users develop and execute application software in a time-sharing Unix/VAX environment in the high-level language "C" to control the manipulator, although the system is not tied to any particular programming language.

2. General Consideration

The control system of a robot manipulator must satisfy a number of basic requirements in order to perform useful tasks: it must be able to communicate with the robot coordinator, receiving instructions and sending back manipulator states, and to interpret the instructions for task execution; it must provide a motion control module in order to position the manipulator where the robot is directed; it must provide compliance when contact is made between the manipulator and the robot environment; it must provide a means by which the action being taken by the manipulator can be modified by sensors in a feedback fashion; finally, it must perform all above operations efficiently to meet the real-time constraints associated with the operations.

Sensors determine the state of a manipulation task in terms of their own coordinate frames. If this information is to be used to control the manipulator, it must be transformed into a common coordinate task frame where given constraints may be applied and information from various sensors integrated to form a best estimate of the task state. This information must in turn be transformed into the manipulator joint coordinates where control of the manipulator is exercised. A key parameter in evaluating the performance of the system is the time delay between a change in some Cartesian coordinate frame and a response at the actuator level, for this time delay determines how effectively sensors can be used to affect the motions. It must either be very slow, such as in welding sensor feedback, or very fast, such as for force or contact feedback. The discussion here is concerned with the latter domain to design a robot force/motion server in which the response of the system is limited only by the manipulator itself and not by the control computer.

The delay of sensor feedback is an accumulated effect due to all the involved processes. It is caused by time spent on interpreting raw data collected by sensors, on transmission of the interpreted result from the sensor to the RFMS, on its effect on Cartesian adjustment to be made by the manipulator, and on the translation of Cartesian adjustment to the joint actuations. While the RFMS can do little to speed up the sensor processing, it can have a strong impact on the time delay in the translation from the Cartesian commands to actuation control signals. the rest of the way.

To minimize the delay due to transmission and create a flexible system, a robot system can be best implemented on a computer network with system components loosely coupled and interacting by way of sending messages. Depending upon the responsibility of a component, an appropriate computer can be provided. The diverse computational requirements and the disparate nature of processing prohibit a system design that tightly couples all its components with a bus structure. The requirements on the flexibility for expansion and modification and on the bandwidth make a local computer network a natural choice. A network structure enables the sensors to monitor the features related to the current motion concurrently with the RFMS, thereby parallelizing all the processings and freeing the RFMS from the time-consuming calculations.

In a robot system with a vision sensor, for example, images are processed at a computational cost much higher than, for example, a conveyor position tracked by a position sensor. There is no reason for the vision system to be connected in a tightly coupled system, where it would share the same data bus with other sensors, causing the system to come to a virtual halt whenever the vision system uses the bus because of the amount of data involved. Special treatment is required on the vision system to achieve real-time performance.

A local computer network provides sufficient speed for the system components to exchange messages. Since messages carry pre-processed geometric information, the their compact size makes them easy to handle by the network. Suppose, for example, that an Ethernet local computer network is used. Its bandwidth is ten mega-bits or 1.25 megabytes per second. In a robot system with *n* components sending messages each of *m* bytes, the worst time delay occurs when all components contends net service at the same time. This amounts to

$$\textit{time delay} = \frac{n \times m}{1.25 \times 10^6} \textit{ seconds.}$$

Most of times, the messages contain descriptions of a coordinate frame represented as a transformation of **o**, **a**, and **p** vectors in floating point numbers. The message size *m* is then 36 bytes and, if system contains $n = 10$ components,

$$\begin{aligned} \textit{time delay} &= \frac{10 \times 36}{1.25 \times 10^3} \, ms \\ &= \frac{10 \times 36}{1.25 \times 10^3} \, ms \\ &\approx 0.228 ms. \end{aligned}$$

Obviously, the time delay of this magnitude is sufficient for the sensors to be closed in the feedback loop controlling the motion of the manipulator, provided the sensor processing itself does not cause much time delay.

3. Identification of the RFMS Processes

The structure of the RFMS breaks down into four levels, as illustrated in Figure 6.1. At the top level, users write application programs to invoke actions by the manipulator in terms of function calls to a system library, in much the same way that RCCL [Hayward and Paul 1984] works; at the next level, information is exchanged between the RFMS and the robot coordinator - the robot coordinator instructs the RFMS by defining the world model and issuing motion/force requests and the RFMS sends the status of the manipulator back to the coordinator; at the third level, the RFMS updates manipulator kinematic states, its Jacobian matrices, coefficients in the dynamics equations, etc; at the bottom level, real-time computation takes place to calculate manipulator set-points and joint actuations.

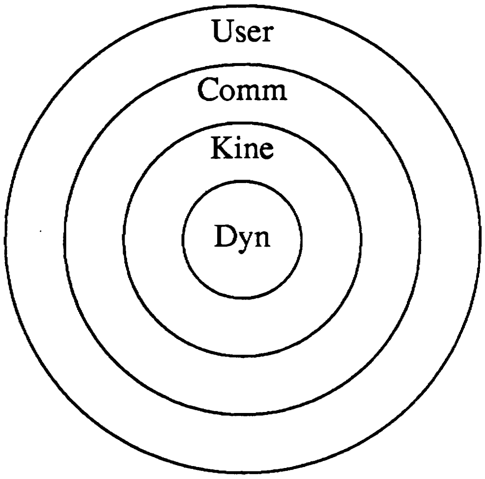


Figure 6.1. Levels of Control in the RFMS

It is important to separate kinematic processes and the dynamic computations in the above hierarchy, for the dynamic processes must maintain a sufficiently high sampling rate, f_s, whereas kinematic processes are related only to manipulator configuration changes. To achieve a stable system, sampling rate of the dynamic process must be at least twice of the system structural frequency. For a manipulator like PUMA 560, the structural frequency is about 20 to 30 Hertz for the wrist joints and, therefore, f_s must be at least 60 to 70 Hertz. For motion to be better behaved, it is often desirable to have a sampling rate three or four times higher than the minimum requirement to over 300 Hertz. However, if this sampling rate is achieved at the price of lowered rate of Cartesian set-point computation, no effective sensor integration can be achieved, as is the case with many industrial robot controllers. On the other hand, by separating kinematic computations from the dynamic computations, the complexity of the dynamic process is lessened with no degraded performance. The update of Jacobian matrix, for example, belongs to the kinematic process; it changes very little when the manipulator executes fine motion at low velocity and it changes fast if the manipulator executes gross motion when the control accuracy is unimportant.

It is also important to use as little pipeline algorithm and as much parallel algorithm as possible, for the pipelined system leads to longer time delay than a paralleled system, although a pipelined system can increase the system throughput. There is little point, for example, in utilizing a processor to perform manipulator joint control input/output, another to perform kinematic transformations, another to convert from accelerations to joint torques as these processes must be pipelined, performed one after the other. This increases the rate at which the calculations may be performed but does nothing to minimize the time delay.

3.1. User Process

The user process executes in a multi-user time-sharing environment to define manipulator tasks by function calls in four general classes: *transform* definitions, *position* definitions, *mode* definitions, and *action requests*.

Coordinate frames representing relationships between objects in the robot environment are defined by *transform* definitions. A transform structure looks like

```
typedef struct trsf {
     VECT n, o, a, p; /* vect is a [x y z] */
     int id;
     int type; /* a variable transform? */
     char *name;
} TRSF, *TRSF_PTR;  /* transform message is TRSF itself */
```

and a call

```
trsf = gentr_const(args);
```

defines a constant transform that remains unchanged unless explicitly redefined, where `args` are the arguments to the function to specify what kind of constant transform to define, such as a pure translation transform or a rotational transform in terms of Euler angles. A call like

```
trsf = gentr_var(args);
```

on the other hand, defines a variable transform that is dependent on external signals such as sensors. If the transform is variable, each time a new sensor observation is obtained, it is redefined in the world model. This mechanism enables a user to modify manipulator motions by sensor observations. All the transforms are stored in a symbol table and their `ids` are indices to the table for later references.

Positions to which a manipulator is to move can be defined by function calls to *position equation* definition.

```
pos = makeposition(leftl, rightl, ···);
```

A position equation is defined as

```
typedef struct posmsg {
    int lfid[MaxLen],    /* left hand side < MaxLen trsfs */
        rtid[MaxLen];    /* right ... */
    int tool,       /* tool transform */
        comply;          /* comply position */
    int id;
    BOOL flip, left, up;     /* configurations */
    char *name;          /* name of the position eq */
} PST, *PST_PTR;
```

There are two ordered lists, `leftl` and `rightl`, of transform identifiers corresponding to the left and right hand sides of a position equation. The transforms used must have been defined prior to the position definition. Other attributes associated with a position equation include configurations of the position, for the position is specified in Cartesian coordinates and there may be more than one set of joint positions that can satisfy the given position. With the configuration specified, a unique set of joint positions can be determined.

A *mode* structure specifies how each motion is to be made, with parameters such as segment time, acceleration time, velocity, compliance specifications, etc., and it is defined in a structure **M** as

```
m=makemode(tseg, tacc, mode, ···);
```

It has the structure:

```
typedef struct {
     int id;
     int mode; /* mode of the motion */
     float tseg;    /* segment time */
     float tacc;    /* acceleration time */
     int cpysel;    /* compliance selection word */
     FORCE force;   /* force to be exerted  */
     int stpsel;    /* stop on force direction word */
     FORCE stpforce; /* stop on force values */
     int dissel;    /* stop on distance selection */
     DIFF dist;     /* distance values */
     float mass;    /* load held */
} MODE, *MODE_PTR;
```

where `FORCE` and `DIFF` are vectors of six floats. One may define a number of modes in a program and associate motion segments with modes. It may be sometimes desirable to execute a motion one way in the beginning of a program and then execute the same motion later in the program with another mode specification. The definition of mode structures and their association with a motion makes the programming of tasks simple and clear.

To request an action by the manipulator, *motion request* is called. Each request is stored in a structure like

```
typedef struct reqmsg{
     int pstid;     /* which position to go to */
     int motid;     /* with what mode */
     int id;
} REQ, *REQ_PTR;
```

A motion request relies on the defined data structures in the first three categories to initiate actions for the manipulator. A request contains two pointers: one to a destination position with the current position being the default initial position, and the second pointer to a mode structure specifying how the motion is to be made. It is called in the form of

```
s=move(pos, mode);
```

Moves are executed sequentially in the order they are requested and each move request returns a sequence number, which can be used to identify moves for motion synchronization. For example, when one wishes to make sure the next move will not start until the current move finishes,

```
waitmove(s);
```

serves for this purpose.

3.2. Communication Process

The communication process enables the RFMS to communicate with the robot system through the network. The robot coordinator issues messages to the RFMS and may request information from the RFMS. The messages from the coordinator contain the definition of the world model in terms of transform and position definitions and move requests. The world model can be further modified by new definitions of transformations. The RFMS broadcast its status back to the coordinator on a regular basis. The received messages are stored in the memory to be processed by another process.

Even though the definition of the world model and motion requests are sent with no real-time constraints, when the sensors extract information from the robot environment, descriptions of interesting features must be provided to the RFMS in real time to facilitate sensor-driven manipulator motions. The real-time constraints on the communication process, when the exchanged information involves sensor observations, require that the messages reach the RFMS within a specified time delay. Tracking the position of a moving object by a vision sensor, for example, requires the observation be transmitted faster than the object moves so that tracking operation will be stable. The speed of communications at this level must be at millisecond level, and, as described in previous section, a local computer network at real-time constraints suffices for this purpose.

The content of a message is contained in a structure `MESS` with the form

```
typedef union {
     TRSF trsf;
     PST pst;
     MOT mot;
     REQ req;
} MESS, *MESS_PTR;
```

and a message can then be defined as

```
typedef struct {
     char type;        /* of the message */
     int id;           /* serial number */
     char used;        /* if the message has been read */
     MESS mess;        /* contents */
} MSG, *MSG_PTR;
```

The field `id` is used as a serial number of the messages and the field `used` identifies if the message has been processed and, with this field, a message will not be mistakenly read twice or destroyed before processed.

3.3. Kinematic Process

The kinematic processes are computed at the rate of change of manipulator configurations. At this level, the RFMS performs computations related to the kinematic states of the system, which are functions of only manipulator configurations and can be computed in background. Failure to perform them fast enough will not lead to system instability, but less accurate control. When manipulator remains stationary, these processes do not have to be updated at all.

Two major computational elements at this level are the computations of Jacobian matrices and of coefficients of dynamics equations. The Jacobian matrix can be symbolically derived as functions of joint positions. It is useful in computing joint torques

τ from a Cartesian force vector $\mathbf{f}_c$ and in any other applications based in Cartesian coordinates. The stiffness control [Salisbury 1981], for example, requires the computation of joint stiffness matrix $\mathbf{K}_\theta$ and the modified hybrid control [Zhang and Paul 1985] requires joint compliance matrix $\mathbf{C}_\theta$, both of which depend on the Jacobian matrix and/or inverse Jacobian matrix. The coefficients in Lagrangian dynamics equations are also functions of joint positions and can be derived symbolically. Their evaluation at the rate of kinematic process is both economical and sufficient.

3.4. Dynamic Process

The processes at the center level are interrupt driven and proceed at the sampling rate of the system with the strictest real-time constraint. Computation in the processes must be finished before the next interrupt comes and the sampling rate must be high enough to assure the smoothness of the motions as well as the stability of the system. The process is given the highest priority within the controller hierarchy. The task of the process consists of a set-point process to compute desired joint positions and a force control process to compute joint actuations.

A set-point process computes the set-point in two stages. First it solves for $\mathbf{T}_6$ from the current position equation P_i and possibly the next position equation P_{i+1} if a transition is necessary. This requires matrix operations such as matrix inversion and multiplication. Obviously, the more complex the position equation, the more expensive this process is computationally. Equation (6.1) functionally defines this first stage:

$$\mathbf{T}_6 = \Gamma(P_i, P_{i+1}, \mathbf{M}, t) \tag{6.1}$$

The second stage of the set-point process solves for the joint positions from the $\mathbf{T}_6$. When the manipulator is simple, this can be performed symbolically, i.e., closed-form solutions exist to express the joint positions as a function of the $\mathbf{T}_6$.

$$\theta_d = \Lambda(T_6, \mathbf{M}, t) \tag{6.2}$$

Once the set-points are computed, the RFMS computes the joint torques by another force control process. It is in this process that dynamics is compensated for and force control algorithms is implemented. A general form for torque for the *i*th joint is:

$$\tau_i = \mathbf{D}(\theta, \dot{\theta}, \ddot{\theta}) + \mathbf{F}(\mathbf{f}_c, \mathbf{J}) + \mathbf{R}(\theta_d, \mathbf{S}, \mathbf{J}) \tag{6.3}$$

where **D** () is dynamics compensation, **F** () is joint bias force due to the Cartesian bias force $\mathbf{f}_c$, and **R** () is the reaction torque due to joint errors $d\theta$ and with the compliance specification **S** taken into consideration.

Equation (6.3) reflects the general approach of all joint based force control methods such as the stiffness method and hybrid method. Note here that all the input parameters to Eq. (6.3) are available from the kinematic process computed in the background and that each function in Eq. (6.3) is a linear function of the input arguments. Therefore, it represents very little computational load and can be executed efficiently.

4. Implementation of the RFMS

Based on the above theoretical analysis and using motion planner in Chapter III and the hybrid control in Chapter V as control strategies, a distributed multi-processor RFMS, illustrated in Figure 6.2, is built with Intel's single board computers, to control a Unimation PUMA 250 manipulator [Unimation 1980]. The real-time constraints on the processes determine their priorities in assigning processors to processes. The fact that the time delay between a change in some Cartesian coordinate frame and a response at the actuator level can be minimized by performing as many calculations in parallel as possible, leads to this architecture in which one processor is used for each joint of the manipulator, one processor is dedicated to supervising the system, the kinematic process is computed at a slow rate in the background on a math processor, a matrix multiplier [Nash 1985] is used to compute matrix-related operations, limiting the real-time process for a joint to the task of converting T_6 to its joint position, adjusting the joint errors if compliance is required, and converting joint accelerations to joint inertia compensation. At this time, however, the matrix multiplier is yet to become available from the manufacturer; it is being simulated as the real-time job on the math processor.

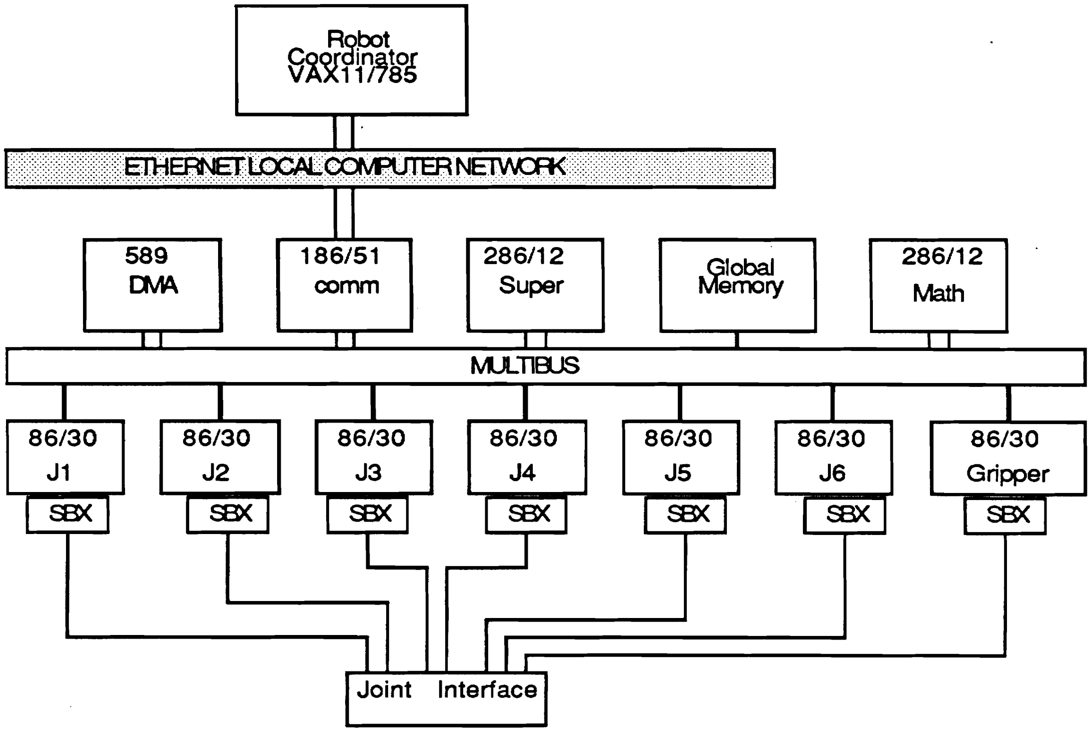

Robot Coordinator VAX11/785
ETHERNET LOCAL COMPUTER NETWORK
589 DMA
186/51 comm
286/12 Super
Global Memory
286/12 Math
MULTIBUS
86/30 J1
86/30 J2
86/30 J3
86/30 J4
86/30 J5
86/30 J6
86/30 Gripper
SBX
SBX
SBX
SBX
SBX
SBX
SBX
Joint Interface

Figure 6.2. The RFMS Implementation

The interface of the RFMS to the coordinator and to sensor systems is provided by the Ethernet [Intel, DEC, and Xerox 1982a] computer network. The RFMS itself is tightly coupled by the Multibus system bus [Intel 1983a]. The Multibus interface is a general purpose system bus structure providing for communication between system components. Memory on one board can be accessed by another through the Multibus; eight interrupt signals can be used to direct actions. Even though any board on the Multibus can become the bus master, Multibus usage by boards other than the supervisor is kept at the minimum in the system to minimize the bus contention. The Multibus system bus can access up to 16 megabytes of system memory. However, the system can be most efficient if the system memory is restricted to a one megabyte page to avoid operations for changing a megabyte page. Given the size of programs in the RFMS, one megabyte is sufficient. The detail of Multibus memory assignment is given by Table 6.1.

Table 6.1. Multibus Address Assignment

Device	*Start Address*	*Ending Address*	*Size*
Super	0x00000	0x1FFFF	128k
Joint 1	0x20000	0x2FFFF	64k
Joint 2	0x30000	0x3FFFF	64k
Joint 3	0x40000	0x4FFFF	64k
Joint 4	0x50000	0x5FFFF	64k
Joint 5	0x60000	0x6FFFF	64k
Joint 6	0x70000	0x7FFFF	64k
Gripper	0x80000	0x8FFFF	64k
Comm	0x90000	0xAFFFF	128k
DMA	0xB0000	0xB3FFF	16k
Mem	0xB4000	0xFBFFF	288k
SDM	0xFC000	0xFFFFF	16k

The Intel single board computers all provide standard Multibus interface for easy interactions among the boards. Joint processors and supervisor as well as the math processor contain 128 kilobytes on-board memory each, which can be chosen by jumper selections to be system memory accessible from the Multibus or local memory

To achieve efficiency, the RFMS does not use an operating system but is driven by interrupts and handshaking operations. The system is programmed in high level language "C" [Kernighan and Ritchie 1978] except for some low-level I/O and interrupt and timer control, which are programmed in *a* 86, a hybrid between Intel's assembly language and VAX assembly language. The programming of SBCs is by way of a cross-compiler using the development system illustrated in Figure 6.3. Programs are written and compiled on a VAX/Unix system and down-loaded to the target system for execution through a serial interface. Arithmetic operations are performed in floating point numbers with Intel's floating point co-processors. Integer arithmetic can be more efficient but gives poor precision; the coding in integers are also difficult for understanding and modifications.

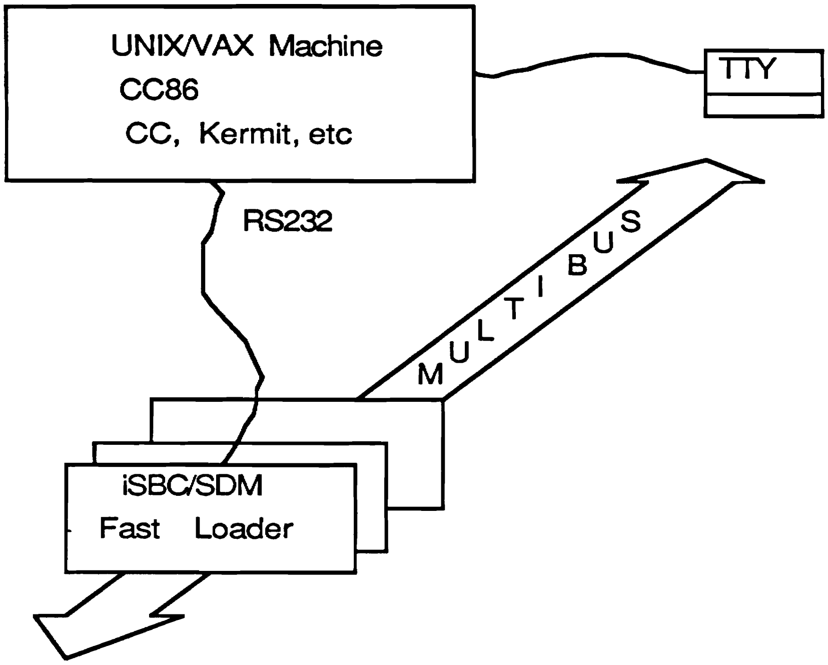


Figure 6.3. Development System

4.1. User Level

The robot coordinator runs on a VAX 11/785 under Unix in a time-sharing environment. Since the intent of the research here is not to construct a dedicated robot manipulator control system, but a flexible force/motion server easily interfaced to and controlled by a robot system, no formal robot control system is specified. There are available to the users a set of primitive functions that provide fundamental control features. Associated with each function call is a message to be sent to the RFMS.

The world model is maintained at this level with transformation, position equation, and motion mode symbol tables for the communication process to identify the redefinitions of the world model. A redefinition of a transformation leads to a modified transformation. A position or motion mode can only be defined once; any attempt to redefine them is considered as an error by the system.

4.2. Ethernet Communication

An Intel iSBC 186/51 [Intel 1984] communication controller in the RFMS facilitates the communication between the robot coordinator and the RFMS. It is connected to the coordinator via an Ethernet transceiver on one end and connected to the RFMS via Multibus on the other. The communication with the coordinator takes place in the form of messages. When a message arrives, the CPU is interrupt by the co-processor 82586 and the message is processed in the interrupt service routine. The routine stores it sequentially in a message list, of which each entry contains a message identifier or index, a message type, a flag for handshaking with the process reading the messages, and the content of the message.

Upon transmission of a message back to the coordinator, the message is packed into a string of characters and sent by a general command,

```
send(dest, buf, size);
```

where the `dest` is the destination address on the network, `buf` contains the message content, and `size` is the size of the message.

4.3. Supervisor

The supervisor of the RFMS maintains the global variables and directs the real-time processes. It runs on an Intel iSBC 86/30 [Intel 1982b] with an 8Mhz 8087 floating point co-processor [Intel 1980]. The iSBC 86/30 contains 128K of dual-port memory, nine levels of interrupt control, two programmable timers, and serial and parallel I/O interface. The 8087 numeric co-processor executes floating point instructions and its instruction set provides for both arithmetic and trigonometric functions, such as tangent and rctangenet.

In the background, the supervisor processes the message list on the Ethernet communication controller and creates its own copy of world model. In real time, it controls the system clock, provides coordination among processors, and computes Eq. (6.3) with the help of the math processor. The memory organization of the RFMS created by the supervisor can be illustrated by Figure 6.4.

The background process reads the messages stored in the iSBC 186/51 and creates the internal data structures in terms of symbol tables and a motion queue. Since there is no variable sharing between the robot coordinator and the RFMS, the RFMS must be able to interpret messages. This can be achieved by setting up symbol tables on the RFMS for various data structures and associating an identifier with each defined structure.

An entry is the transformation symbol table is defined by a structure

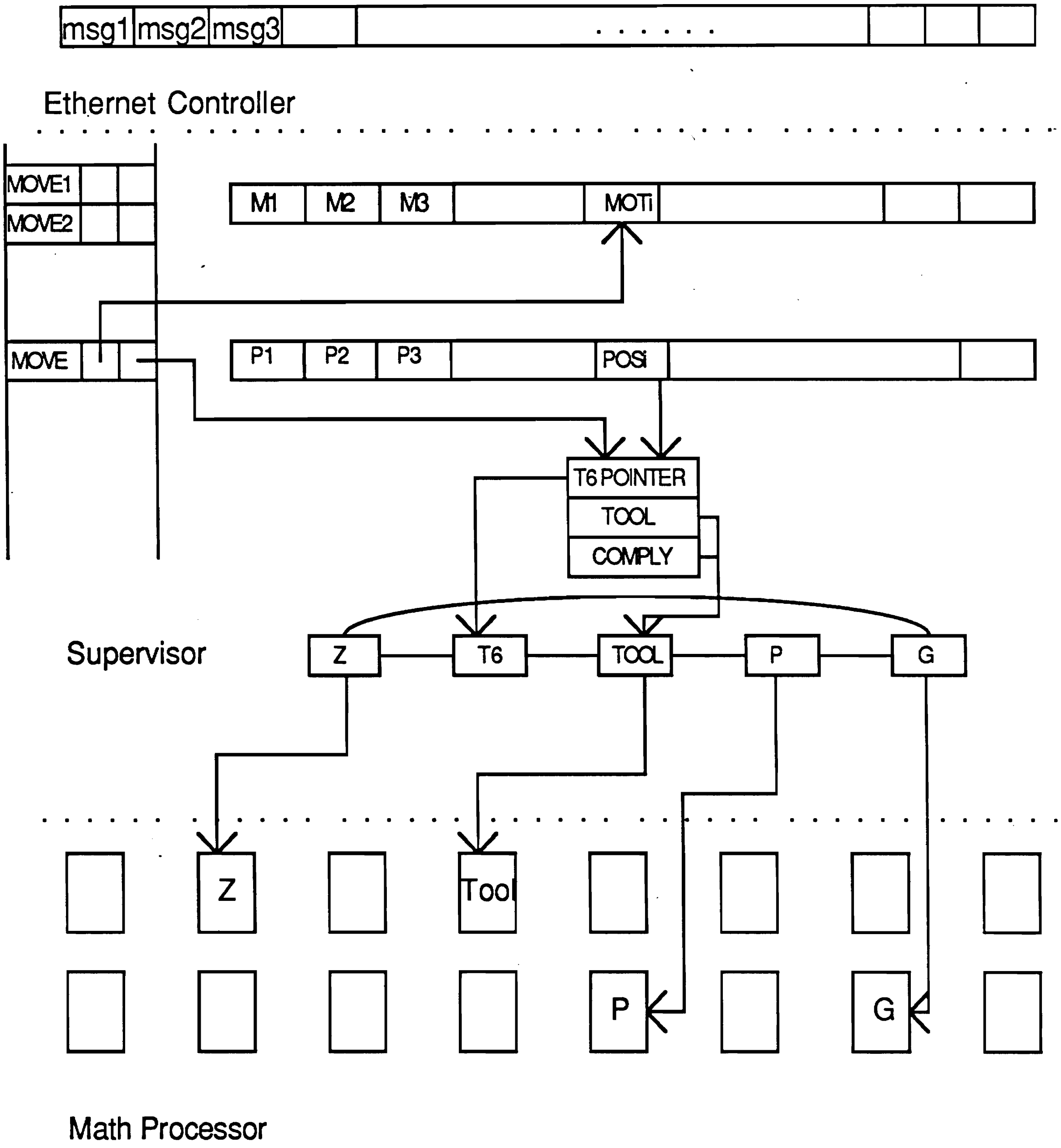

msg1
msg2
msg3
. . . .
Ethernet Controller
MOVE1
MOVE2
M1
M2
M3
MOTi
MOVE
P1
P2
P3
POSi
T6 POINTER
TOOL
COMPLY
Supervisor
Z
T6
TOOL
P
G
Z
Tool
P
G
Math Processor

Figure 6.4. Memory Organization

```
typedef struct trsf {
     int id;
     NOAP *noap;          /* pointer to transform content */
     struct trsf *invs;       /* inverse of the transform */
     char invp;           /* inverse predicate */
     char valid;          /* whether the info is up to date */
} TRSF, *TRSF_PTR;
```

where `NOAP` is

```
typedef struct {
     VECT n, o, a, p;
} NOAP, *NOAP_PTR;
```

`id` is the identifier of the transformation, `invs` points to the inverse of the transformation, and an `invp` serves as an inverse transformation predicate. `noap` points to the contents of the transformation, which are stored in the matrix multiplier memory where the matrix operations are performed; `noap` is implicitly a pointer to memory on another board. This arrangement avoids the need to move the contents of transformations, since only the matrix multiplier needs to have access to the contents.

A position equation references transforms in its definition. it can find the transformations associated with the position equation from the transform symbol table. When a position equation is processed, the two lists of transformations are first looked up in the transform symbol tables. A ring structure [Paul 1981] illustrated in Figure 6.5 is then created for each position and the pointer to the structure is stored in a position symbol table. A mode record is treated similarly in a mode symbol table. Finally, an action request is queued in the action queue after its two pointers to a position equation and to a mode record are located in the respective symbol tables.

The real-time process on the supervisor initiates the computation of the next set-point upon a real-time clock interrupt from the programmed timer every sample period. The algorithm used is that in [Paul and Zhang 1985]. The supervisor first determines how the next Cartesian set-point, $\mathbf{T}_6$, should be computed and requests the math processor to compute the all the matrix inversion and multiplications. It then requests the data channel to communicate $\mathbf{T}_6$ to the joint processors to compute the joint coordinates. The specific computation performed during each period depends upon the *state* of the manipulator or its trajectory. This process must finish before the next interrupt comes when another set-point must to be generated.

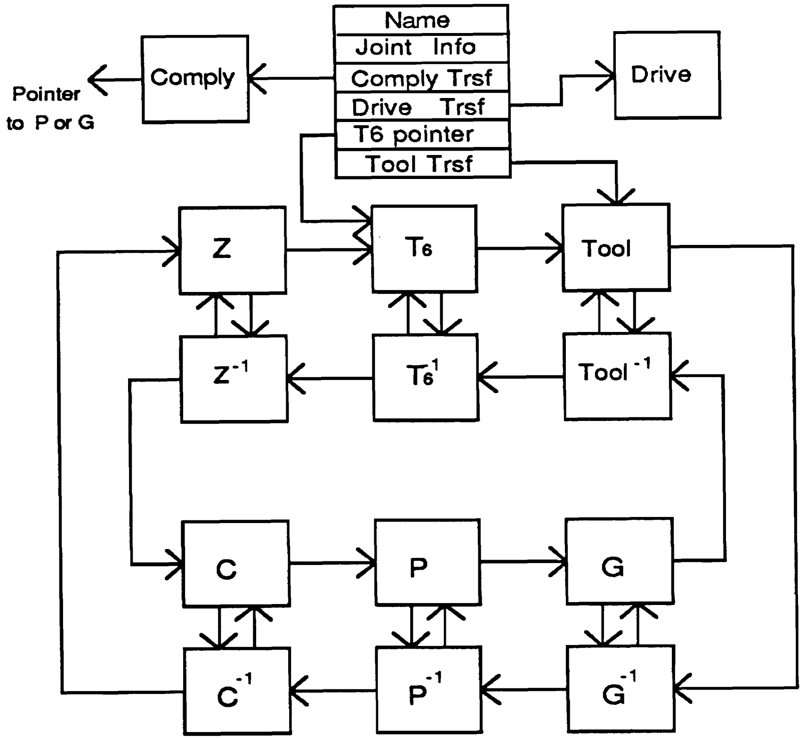

Name
Joint Info
Comply Trsf
Drive Trsf
T6 pointer
Tool Trsf
Comply
Pointer to P or G
Drive
Z
T6
Tool
Z-1
T6-1
Tool-1
C
P
G
C-1
P-1
G-1

Figure 6.5. Position Ring Structure

The math processor performing matrix operations is simulated now by an Intel iSBC 286 [Intel 1985], which will eventually be replaced by a matrix multiplier device [Nash 1985], a systolic array processor, to compute the matrix operations for the supervisor at a speed needed to write and fetch data. Currently a math processor simulates matrix multiplier and in each sampling period, when the matrix multiplications are needed, the supervisor generates a pointer list to those matrices to be multiplied, interrupts the math processor, and then continues with other operations. Upon completion of matrix multiplications, the math process interrupts the supervisor.

4.4. Joint Processors

Each joint is equipped with an iSBC 86/30 and an 8087 numeric processor. 64 kilobytes of its memory is configured global for interprocess communication. Joint processes run in parallel to control individual joints as described by Eq.(6.2) and Eq. (6.3). Working as slave processors to the supervisor, each of the joint processors computes its joint trajectory. The only process on a joint processor is real-time and interrupt driven. Its interaction with the supervisor can be described by table 6.2.

Table 6.2. Supervisor-Joint Interaction

Supervisor	*Joint i*
Send $\mathbf{T}_6$	background if any
interrupt all joints	acknowledge interrupt
start computing next $\mathbf{T}_6$	compute inverse kinematics and $d\theta_i$
acknowledge interrupt	interrupt supervisor to signal finish
read $d\boldsymbol{\theta}$ and send it back to all joints	wait
resume interrupted job	compute $d\theta_{ri}$, if necessary, and $\ddot{\theta}$
acknowledge interrupt	interrupt supervisor
read $\ddot{\boldsymbol{\theta}}$ and send it back to all joints	wait
resume interrupted job	compute τ_i and servo the joint

At the beginning of a sampling period, it reads the current desired Cartesian position computed by the supervisor and solves for the joint position. Since the ith joint solution requires sines and cosines of prior i-1 joints in general and this would cause considerable time delay if joints wait for solutions, one can make use of the sines and cosines of joint variables computed in the last sample period so that all joints start computing simultaneously. During a transition, however, a joint process computes the coefficients of the transition polynomial and obtains solution by evaluating the polynomial. The *state* variable of the supervisor dictates the action of the joint processes.

The force control process is executed next to compute joint torque as in Eq. (6.3). Information such as the Jacobian matrix and dynamics equation coefficients are broadcast to the joints when they are updated by the kinematic processes, with the *i*th row of the joint compliance matrix and of the inertial matrix passed to the *i*th joint. On the other the joint errors need to be exchanged among joints in real time for hybrid control. Noting that the joint processors are started to compute joint set-points simultaneously by the real-time clock on the supervisor, the joint with the most computation must finish the last, at which time it can interrupt the supervisor to exchange the joint errors and be sure that all other joints have finished. The same argument holds for the exchange of joint accelerations needed for dynamics compensation. Once the real error is computed and joint accelerations obtained, Eq. (5.10a) can be evaluated to generate joint reaction torques as in Eq. (5.10a) and Eq. (6.3).

The low level interface to the joint is achieved through a specially designed hardware, iSBX multimodule board [Zhang 1986], that is attached to the SBX connector on each host 86/30. It provides the joint encoder interface and A/D and D/A converters. The board employs two digital to analog converters, two analog to digital converters, and an incremental encoder circuit. The first DAC outputs motor current, while the second allows one to specify a force set point to the joint. The two ADCs reads the joint motor current and joint velocity, respectively. The incremental encoder tracks the position of the joint by observing the waveforms generated by the joint encoder. At the end of each sample period, the computed torque is converted to an equivalent joint current value through a PID control as in Eq. (5.11) and sent to the amplifier circuits to drive the joint motors.

4.5. Math Process

The math processor performs matrix multiplications in real time and computes the kinematic process in the background. It is implemented on an Intel iSBC 286/12 While the iSBC 286/12 has one megabyte on-board dual-port memory, 64K is defined as Multibus memory. The SBC contains a programmable interrupt controller, programmable timers, Multibus interface, and parallel and serial I/O interface, and the numeric co-processor 80287 [Intel 1985] equivalent to the 8087.

The matrix multiplication starts with an interrupt from the supervisor. The list of pointers to the matrices to be multiplied is stored in the Multibus memory at a fixed location and terminated by a null pointer. The matrix multiplication function is executed as the interrupt service routine. The result of the multiplication is stored in the Multibus memory, the pointer to which is written to another fixed location for the supervisor to read. The math processor then interrupts the supervisor to signal the finish.

The process performs dynamics and Jacobian matrix updates in background continuously. For the application of hybrid control, the joint compliance matrix $\mathbf{C}_\theta$ is also updated. For a typical robot manipulator like PUMA, each cycle of dynamics update takes roughly 21 *ms* and $\mathbf{C}_\theta$ update about 14 *ms* [Zhang 1986]; therefore, the rate of update for the background process is about 30 Hertz.

4.6. Direct Memory Access

A DMA board iSBC 589 [Intel 1982c] is added to the system to speed up the data transfer operations. It enhances system performance by helping the supervisor with the data transfers. Because the supervisor and the joint processors exchange information frequently in their real-time processes, this device considerably reduces the time for global memory read/write. The iSBC 589 provides an ideal solution to such a system in which exchanged information is of fixed format and source and destination are known a priori. To invoke a DMA action, the supervisor simply specifies one of the pre-constructed parameter blocks containing source and destination addresses and the size of the data block, then it may proceed to its next instruction, leaving the I/O to the DMA controller. Further, the speed of transfer by iSBC 589 can be up to one megabytes per second, much faster than that of an 86/30. At the time of initialization, destination and source addresses for different data transfers are stored in the parameter blocks. In real-time, the supervisor simply issues a wake-up byte to the parameter block corresponding to the desired operations.

4.7. Performance Evaluation

The real-time performance of the system depends on the control strategies. For this experiment, the motion trajectory generator in Chapter III and the hybrid controller controller in Chapter V are implemented as the control strategies. As indicated earlier, the worst case of the TG occurs at the starting point of a transition to Cartesian motion, when $\mathbf{T}_6$ evaluation, two inverse kinematics solutions, and a drive transformation are to be computed.

The Cartesian set-point generation described by Eq. (6.2) involves the $\mathbf{T}_6$ evaluation, which requires matrix multiplications. The matrix operations can be performed fast if the matrix multiplier is available; the time spent is that for transferring data to the bus and for reading the results back. However, before such a device is available, the math processor is to perform the operations. The computational complexity of Eq. (6.2) depends on the number of transformations in the position equations. Each matrix multiplication requires 33 multiplies and 24 adds and each matrix inversion requires 15 multiplies and nine adds. In an average case where there are four transformations in the position equation like

$$\mathbf{T}_6\ \mathbf{A} = \mathbf{B}\ \mathbf{C}, \tag{6.5}$$

a total of 81 multiplies and 57 adds are needed. This amount of computation corresponds to five *ms* on an 8087 floating point processor. A drive transform computation requires 19 multiplies, 12 adds, three arctangent calls, two square root calls, and two sine/cosine functions. This corresponds to about 3.27 *ms* on an 8087.

The inverse kinematics, Eq. (6.3), are computed in parallel on all the joint processors. For this implementation on PUMA 250 manipulator, using the kinematic solutions given in Appendix A, the worst case occurs on joint two and six, for which 1.3 *ms* is needed for one inverse kinematics solution. The final torque computation described by Eq. (6.4) requires additional 1.4 *ms*. Therefore, the total amount of computation for each joint is about 4 *ms*.

Assume that the matrix multiplier is available. If the processes on supervisor and joint processors are to be computed in serial, i.e., the supervisor computes $\mathbf{T}_6$ first then gives it to joints to compute joint actuations in the same sampling period, the time periods of over 7 *ms* is needed, corresponding to a servo rate of 130 to 140 Hertz. The $\mathbf{T}_6$, however, can be pipelined, as in Figure 6.6, to achieve a higher servo rate.

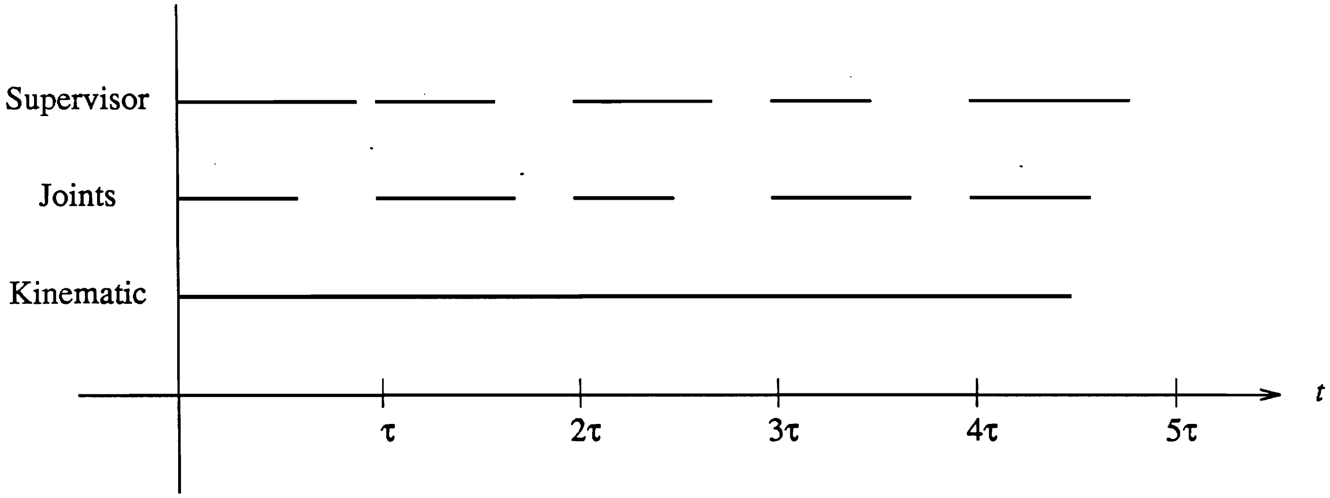


Figure 6.6. Process Scheduling

The Cartesian set-point $\mathbf{T}_6$ computed in ith sampling period is used in $(i+1)$th sampling period, while the math process continues to execute in parallel with and providing information to the real-time processes. Joint processors start computing simultaneously with the supervisor in each sampling period, in which case the servo rate of the system depends on the longer of the supervisor process and the joint processes. Using the numbers above, a 250 Hertz sampling rate can be obtained. This represents a considerable improvement over all the industrial robot controllers, where the Cartesian update rate is almost ten times slower.

5. Conclusion

This Chapter discusses the design of a robot manipulator controller with sufficient computational bandwidth and precision so that the manipulator performance limits the performance of the system not the controller. The system provides for both position and force control and is not dedicated to a programming language but is specified in terms of network message formats. An Ethernet interface is provided so that the system may be directly interfaced with many other sensors and robot planning and control systems in a very simple manner. The high computational bandwidth facilitates motions controlled by sensors in real time.

Off-the-shelf components are used in the implementation to achieve the system economically. All coding is in the C language to facilitate understanding and future modification of the system.

CHAPTER VII
CONCLUSION AND FUTURE WORK

1. Summary

For robots to be truly intelligent systems capable of functioning independently, handling uncertainties, and performing useful tasks, they must be able to react to changes taking place in their environment through the use of sensors, which help them understand and reason about the world. Further, in order to perform, they must also be able to take actions effectively based on the task specification and interpretation of sensor observations. Significant process has been made in the past few decades in various areas of the robotics, in sensing techniques, in machine reasoning, and in robot manipulation. The various disciplines, however, are yet to be integrated to create useful systems. This represents a next major undertaking for robotics researchers.

The research described in this thesis attempts as the first step at the problem of a complete robot system construction. A robot force/motion server is designed and constructed for the robot system, to provide motion control and compliance and to facilitate real-time sensor driven motions. The motion control strategy provides essential forms of motions; the force control strategy enables a manipulator to perform tasks constrained by the task geometry such as an assembly. Both control strategies are efficient for real-time control of manipulators with a sufficiently high Cartesian set-point update rate. The use of the simplified dynamics model improves the performance of the system and provides a realistic and efficient way of accounting for the manipulator dynamics. A multi-processor system with an adequate system throughput is built to perform computations necessary for the control strategies, although it is not restricted to any particular algorithm, but rather provides a general computing system on which different control techniques can be applied. The Ethernet interface built into the server provides the server with real-time sensory feedback and makes it convenient to drive the RFMS from other computers.

2. Future Work

There are a number of areas of the RFMS that can be improved, based on the development described in this writting. In a trajectory planner, transitions sometimes may have to be performed in the Cartesian space for the operations, such as seam welding, where precise Cartesian positions are demanded at all times including during the transitions. Since the trajectory generator introduced and implemented here performs all transitions in the joint space to obtain efficiency of operation, the Cartesian trajectory during a transition may deviate beyond that which can be tolerated. In order to eliminate the problem, transition could be re-formed in Cartesian space. In order to retain the simple formulation offered by the current method, one could exclusively perform transitions in Cartesian space regardless of the motion of the next segment. This requires more computations but leads to accurate Cartesian path control.

The modified hybrid control method in Chapter V is both theoretically correct and computationally efficient; however, the control performance is limited by the inabilities of the joints to predict direction of the velocity for friction compensation. As a result, the force control sensitivity of the manipulator is that of the joint friction, which is unfortunately larger for all industrial robot manipulators than usually required by an assembly task. To solve the problem, one can design manipulators with low joint friction such as direct-drive arms [Asada 1984]; one can also design joint feedback torque controllers to reduce the joint friction to the minimum [Wu and Paul 1980] [Luh, Fisher, and Paul 1980] [Pfeffer, Khatib, and Hake 1986]. All of the methods, however, involve considerable additional remodeling of the manipulator. A more economical approach is to close the Cartesian force control loop using a Cartesian force sensor and an improved performance can be expected [Craig and Raibert 1979].

To make use of the RFMS effectively, a complete robot programming system must be designed. The sensors and the RFMS are integrated into a coherent system, where the system components function in parallel to contribute to the system with knowledge and capability in their own domain of expertises. A system of such a complex structure cannot be constructed on a single-processor system and unpredictable real-time interactions take place among the robot sub-systems in performing a task. Research should be directed into task level programming systems with capabilities beyond those that programming languages provide, if robots are to become useful in general.

LIST OF REFERENCES

LIST OF REFERENCES

Angeles, J. 1985. "On the Numerical Solution of the Inverse Kinematic Problem," *Int. J. Robotics Res.*, Vol. 4, No. 2, pp. 21 – 37.

Asada, H. and Joucef-Tomi, K. 1984. "Analysis and Design of a Direct-drive Arm with a Five-Bar-Link Parallel Drive Mechanism," *ASME J. Dynamics Sys., Measurements, and Control*, Vol. 106.

Backes, P.G. 1984. "Real Time Control with Adaptive Manipulator Control Schemes," M.S. Thesis, School of Mechanical Engineering, Purdue University.

Bejczy, A.K. 1974. "Robot Arm Dynamics and Control," NASA - JPL Technical Memorandum, 33-669.

Bejczy, A.K. and Lee, S. 1983. "Robot Arm Dynamics Model Reduction for Control," *Proc. IEEE Conf. on Decision and Control.*

Brady, M., Hollerbach, J.M., Johnson, T.L., Lozano-Perez, T., and Mason, M.T. 1982. *Robot Motion.* Cambridge: MIT Press.

Burdick, J. and Khatib, O. 1984. "Force Control Implementation in COSMOS," *Intelligent Task Automation Interim Technical Report*, 1/16 - 4/15, pp. 6-9 – 6-12.

Giralt, G. 1984. "Research Trends in Decisional and Multisensory Aspects of Third Generation Robots," *Proc. Second International Symposium of Robotics Research*, pp. 446 – 455.

Goertz, R.C. 1952. "Fundamentals of General Purpose Remote Manipulator," *Nucleonics*, Vol. 10, No. 11, pp. 36 – 42.

Hartenburg, R.S. and Denavit, J. 1964. *Kinematic Synthesis of Linkages.* McGraw Hill.

Hayward, V. and Paul, R.P. 1984. "Introduction to RCCL: A Robot Control C Library," *Proc. IEEE Conf. on Robotics and Automation*, pp. 293 – 297.

Hollerbach, J.M. 1980. "A Recursive Formulation of Lagrangian Manipulator Dynamics," *IEEE Trans. Systems, Man, Cybernetics,* Vol. 10, pp. 730 – 739.

Inoue, H. 1974. "Force Feedback in Precise Assembly Tasks," Artificial Intelligence Laboratory, MIT, AIM-308.

Intel, 1980. "iSBC 337 Multimodule Numeric Data Processor Hardware Reference Manual," 142887-001, Intel Corporation.

Intel, DEC, and Xerox, 1982a. "Ethernet Data Link and Physical Layer Specification, Version 2.0," Intel, DEC, and Xerox.

Intel, 1982b. "iSBC 86/14 and iSBC 86/30 Single Board Computer Hardware Reference Manual," 14404-002, Intel Corporation.

Intel, 1982c. "iSBC 589 Intelligent DMA Controller Board Hardware Reference Manual," 142996-002, Intel Corporation.

Intel, 1983a. "MULTIBUS Architecture Reference Book," 210883-002, Intel Corporation.

Intel, 1983b. "iSDM 86 System Debug Monitor Reference Manual," 146165-001, Intel Corporation.

Intel, 1984. "iSBC 186/51 COMMputer Board Hardware Reference Manual," 122136-002, Intel Corporation.

Intel, 1985. "iSBC 286/12 Single Board Computer Hardware Reference Manual," 147533-001, Intel Corporation.

Izaguirre, A. and Paul, R.P. 1985. "Computation of the Inertial and Gravitational Coefficients of the Dynamics," *Proc. IEEE Conf. on Robotics and Automation*, pp. 1024 – 1032.

Kahn, W.E. 1969. "The Near-minimum-time Control of Open Loop Articulated Kinematic Chains," Ph.D. Thesis, Computer Science Department, Stanford University.

Kasahara, H. and Narita, S. 1985. "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System," *IEEE J. Robotics and Automation*, Vol. 2, RA-1, pp. 104 – 113.

Kernighan, B.W. and Ritchie, D.M. 1978. *The C Programming Language.* Prentice-Hall, Inc.

Khatib, O. 1985. "The Operational Space Formulation in the Analysis, Design, and Control of Robot Manipulators," *The Third Internation Symposium of Robotics Research*, pp. 103 – 110.

Kriegman, D.J., Siegel, D.M., Narasimhan, S., Hollerbach, J.M., and Gerpheide, G.E. 1985. "Computational Architecture for the Utah/MIT Hand," *Proc. IEEE Conf. on Robotics and Automation*, vol.1, pp. 918 – 924.

Lieberman, M. and Wesley, M. 1976. "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *Proc. Robots VI, Detroit*, pp. 392 – 406

Lord, 1983. "ROBOWRIST Remote Center Compliance Devices," PC-8034b, Lord Industrial Products, Lord Corporation.

Luh, J.Y.S., Walker, M.W. and Paul, R.P. 1980. "On-line Computational Scheme for Mechanical Manipulators," *J. Dynamic Systems, Measurement, Control*, Vol. 102, pp. 69 – 76.

Luh, J.Y.S., Fisher, W.B., and Paul, R.P. 1983. "Joint Torque Control by Direct Feedback for Industrial Robots," *IEEE Trans. Automatic Control,* Vol. AC-28, No. 2.

Maples, J. and Goldman, R. 1984. "Progress on Implementing a High-Speed Salisbury Stiffness Controller," *Intelligent Task Automation Interim Technical Report*, 1/16 - 4/15, pp. 6-13 – 6-28.

Mason, M.T. 1979. "Compliance and Force Control for Computer Controlled Manipulators," Artificial Intelligence Laboratory, MIT, AIM-515.

Nash, J.G. 1985. "A Systolic/Cellular Computer Architecture for Linear Algebraic Operations," *Proc. IEEE Conf. on Robotics and Automation*, pp. 779 – 784.

Nigam, R. and Lee, C.S.G. 1985. "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators," *IEEE J. Robotics and Automation*, Vol. 2, RA-1, pp. 173 – 182.

Paul, R.P. 1972. "Modeling, Trajectory Calculation, and Servoing of a Computer Controlled Arm ," AIM 177, Artificial Intelligence Laboratory, Stanford University.

Paul, R.P. and Shimano, B. 1976. "Compliance and Control," *Proc. 1976 Joint Automatic Control Conf.*, pp. 694 – 699.

Paul, R.P. 1981. *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge: MIT Press.

Paul, R.P. 1983. ''The Computational Requirements of Second Generation Robots", *Japan Robotics Society.*

Paul, R.P., Ma, R., and Zhang, H. 1983a. "Dynamics of Puma Manipulator," *Proc. American Control Conf.*, pp. 491 – 496.

Paul, R.P. and Stevenson, C.N. 1983b. "Kinematics of Robot Wrists," *Int. J. Robotics Res.,*, Vol. 2, No. 1, pp. 31 – 38.

Paul, R.P. and Zhang, H. 1985. "Robot Motion Trajectory Specification and Generation," *Proc. Second International Symposium of Robotics Research*, pp. 373 – 380.

Paul, R.P. and Zhang, H. 1986. "Design and Implementation of a Robot Force/Motion Server," *Proc. IEEE Conf. Robotics and Automation*, pp. 1878 – 1883.

Paul, R.P. and Durrant-Whyte, H.F. 1986. "A Robust, Distributed Sensor and Actuation Robot Control System," *Proc. Third International Symposium of Robotics Research.*

Paul, R.P. and Zhang, H. 1986. "Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation," *Int. J. Robotics Res.: Special Issue on Manipulator Kinematics.*

Pfeffer, L., Khatib, O., and Hake, J. 1986. "Joint Torque Sensory Feedback in the Control of a PUMA Manipulator ," *Proc. IEEE Conf. Robotics and Automation*, pp. 1878 – 1883.

Pieper, D.L. 1968. "The Kinematics of Manipulators Under Computer Control," AIM-72, Artificial Intelligence Laboratory, Stanford University.

Wu, C.H. and Paul, R.P. 1980. "Manipulator Compliance Based on Joint Torque Control," *Proc. 19th IEEE Conf. Decision and Control*, pp. 84 – 88.

Raibert, M.H. and Craig, J.J. 1981, "Hybrid Position/Force Control of Manipulators," *J. Energy Resources Technology*, Vol. 103, pp. 126 – 133.

Salisbury, J.K. 1980. "Active Stiffness Control of A Robot Manipulator in Cartesian Coordinates," *IEEE Conf. Decision and Control,* Albuquerque, New Mexico.

Shimano, B.E., Geschke, C.C., and Spalding, C.H. 1984. "VAL-II, A New Robot Control System For Automatic Manufacturing," *Proc. IEEE Conf. on Robotics and Automation*, pp. 278 – 291.

Silver, D. 1982. "On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators," *nt. J. Robotics Research*, Vol. 1, No. 2, pp. 60 – 70.

Taylor, R.H. 1979. "Planning and Execution of Straight-Line Manipulator Trajectories," *IBM J. Research and Development*, Vol. 23, pp. 424 – 436.

Taylor, R.H., Korein, J.U., Maier, G.E., and Durfee, L.F. 1985. "A General Purpose Control Architecture for Programmable Automation Research," *IBM Research Report*, RC 11416 (#51345).

Turner,T.L., Craig, J.J., and Gruver, W.A. 1986. "A Microprocessor Architecture for Advanced Robot Control,"

Uicker, J.J. 1966. "Dynamic Force Analysis of Spatial Linkages," *Mechanisms Conference.*

Unimation Inc. 1980. "Unimation Series 250 Robot Installation and Operation Guide", Unimation Inc.

Unimation Inc. 1982. "Breaking away from VAL or How to Use Your PUMA without Using VAL", Unimation Inc.

Whitney, D.E. 1972. "The Mathematics of Coordinated Control of Prostheses and Manipulators," *J. Dynamic Systems, Measurement, Control*, pp. 303 – 309.

Whitney, D.E. 1982. "Quasi-static Assembly of Compliantly Supported Rigid Parts," *J. Dynamic Systems, Measurement, Control*, Vol. 104, pp. 65 – 77.

Zhang, H. and Paul, R.P. 1985. "Hybrid Control of Robot Manipulators," *Proc. IEEE Conf. on Robotics and Automation*, vol.1, pp. 602 – 607.

Zhang, H. 1986. "Progress Report on the Implementation of the RFMS", Department of Computer and Information Science, University of Pennsylvania.

APPENDICES

Appendix A

Kinematics of PUMA 250 Manipulator

A Matrices

PUMA 260 link and joint parameters are given in Table A1.

Table A1. PUMA 260 Link and Joint Parameters

link	α	a	d
1	90°	0	0
2	0°	a_2	d_2
3	–90°	0	0
4	90°	0	d_4
5	–90°	0	0
6	0°	0	0

$a_2 = 203.2$mm, $d_2 = 126.23$mm, and $d_4 = 203.2$mm. The offset from the T_6 origin to the mounting flange d_6 is 55.4mm long.

With the link and joint parameters, A matrices are obtained by Eq. (2.4) as:

$$A_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_5 = \begin{bmatrix} C_5 & 0 & -S_5 & 0 \\ S_5 & 0 & C_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_3 = \begin{bmatrix} C_3 & 0 & -S_3 & 0 \\ S_3 & 0 & C_3 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(A.1)}$$

$$A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$S_i = \sin(\theta_i) \quad C_i = \cos(\theta_i)$$

Replace A_2 and A_3 with the product A_{23}:

$$A_{23} = \begin{bmatrix} C_{23} & 0 & -S_{23} & C_2a_2 \\ S_{23} & 0 & C_{23} & S_2a_2 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

where

$$S_{23} = \sin(\theta_2+\theta_3) \quad C_{23} = \cos(\theta_2+\theta_3)$$

Direct Kinematics

The position and orientation of the manipulator are given by the product of the A matrices

$$T_6 = A_1\ A_{23}\ A_4\ A_5\ A_6 \tag{A.3}$$

Define U_i as follows:

$$\begin{aligned} U_6 &= A_6 \\ U_5 &= A_5\ U_6 \\ U_4 &= A_4\ U_5 \\ U_3 &= A_3\ U_4 \\ U_2 &= A_{23}\ U_4 \\ U_1 &= A_1\ U_2 \end{aligned} \tag{A.4}$$

Thus

$$\mathbf{U}_6 = \mathbf{A}_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

Premultiplying by $\mathbf{A}_5$, $\mathbf{U}_5$ is obtained as

$$\mathbf{U}_5 = \begin{bmatrix} C_5C_6 & -C_5S_6 & -S_5 & 0 \\ S_5C_6 & -S_5S_6 & C_5 & 0 \\ -S_6 & -C_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.6}$$

Four local variables are created to represent the four expressions present in $\mathbf{U}_5$

$$\begin{aligned} U_{511} &= C_5C_6 \\ U_{512} &= -C_5S_6 \\ U_{521} &= S_5C_6 \\ U_{522} &= -S_5S_6 \end{aligned} \tag{A.7}$$

and then proceed to obtain $\mathbf{U}_4$ by premultiplying by $\mathbf{A}_4$

$$\mathbf{U}_4 = \begin{bmatrix} C_4U_{511}-S_4S_6 & C_4U_{512}-S_4C_6 & -C_4S_5 & 0 \\ S_4U_{511}+C_4S_6 & S_4U_{512}+C_4C_6 & -S_4S_5 & 0 \\ U_{521} & U_{522} & C_5 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.8}$$

Assign local variables:

$$\begin{aligned} U_{411} &= C_4U_{511}-S_4S_6 \\ U_{412} &= C_4U_{512}-S_4C_6 \\ U_{413} &= -C_4S_5 \\ U_{421} &= S_4U_{511}+C_4S_6 \\ U_{422} &= S_4U_{512}+C_4C_6 \\ U_{423} &= -S_4S_5 \end{aligned} \tag{A.9}$$

and premultiply by $\mathbf{A}_3$ to obtain $\mathbf{U}_3$.

$$\mathbf{U}_3 = \begin{bmatrix} C_3U_{411}-S_3U_{521} & C_3U_{412}-S_3U_{522} & C_3U_{413}-S_3C_5 & -S_3d_4 \\ S_3U_{411}+C_3U_{521} & S_3U_{412}+C_3U_{522} & S_3U_{413}+C_3C_5 & C_3d_4 \\ -U_{421} & -U_{422} & -U_{423} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.10}$$

New local variables are assigned:

$$\begin{aligned} U_{311} &= C_3U_{411}-S_3U_{521} \\ U_{312} &= C_3U_{412}-S_3U_{522} \\ U_{313} &= C_3U_{413}-S_3C_5 \\ U_{314} &= -S_3d_4 \\ U_{321} &= S_3U_{411}+C_3U_{521} \\ U_{322} &= S_3U_{412}+C_3U_{522} \\ U_{323} &= S_3U_{413}+C_3C_5 \\ U_{324} &= C_3d_4 \end{aligned} \tag{A.11}$$

As joints 2 and 3 are parallel, $\mathbf{U}_4$ is premultiplied by $\mathbf{A}_{23}$ to obtain $\mathbf{U}_2$

$$\mathbf{U}_2 = \begin{bmatrix} C_{23}U_{411}-S_{23}U_{521} & C_{23}U_{412}-S_{23}U_{522} & C_{23}U_{413}-S_{23}C_5 & -S_{23}d_4+C_2a_2 \\ S_{23}U_{411}+C_{23}U_{521} & S_{23}U_{412}+C_{23}U_{522} & S_{23}U_{413}+C_{23}C_5 & C_{23}d_4+S_2a_2 \\ -U_{421} & -U_{422} & -U_{423} & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.12}$$

There are eight local variables in $\mathbf{U}_2$

$$\begin{aligned} U_{211} &= C_{23}U_{411}-S_{23}U_{521} \\ U_{212} &= C_{23}U_{412}-S_{23}U_{522} \\ U_{213} &= C_{23}U_{413}-S_{23}C_5 \\ U_{214} &= -S_{23}d_4+C_2a_2 \\ U_{221} &= S_{23}U_{411}+C_{23}U_{521} \\ U_{222} &= S_{23}U_{412}+C_{23}U_{522} \\ U_{223} &= S_{23}U_{413}+C_{23}C_5 \\ U_{224} &= C_{23}d_4+S_2a_2 \end{aligned} \tag{A.13}$$

Finally, premultiplying by $\mathbf{A}_1$, one obtains $\mathbf{U}_1 = \mathbf{T}_6$ representing the position and orientation of the end of the manipulator.

$$\mathbf{U}_1 = \mathbf{T}_6 = \begin{bmatrix} C_1U_{221}-S_1U_{421} & C_1U_{212}-S_1U_{422} & C_1U_{213}-S_1U_{423} & C_1U_{214}+S_1d_2 \\ S_1U_{211}+C_1U_{421} & S_1U_{212}+C_1U_{422} & S_1U_{213}+C_1U_{423} & S_1U_{214}-C_1d_2 \\ U_{221} & U_{222} & U_{223} & U_{224} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.14}$$

The elements of $\mathbf{U}_1 = T_6$ are identified as

$$\begin{array}{lll} o_x = C_1U_{212}-S_1U_{422} & a_x = C_1U_{213}-S_1U_{423} & p_x = C_1U_{214}+S_1d_2 \\ o_y = S_1U_{212}+C_1U_{422} & a_y = S_1U_{213}+C_1U_{423} & p_y = S_1U_{214}-C_1d_2 \\ o_z = U_{222} & a_z = U_{223} & p_z = U_{224} \end{array} \tag{A.15}$$

A procedure to evaluate the position and orientation of the end of the manipulator is the local variable assignments.

$$\begin{aligned} U_{512} &= -C_5S_6 \\ U_{522} &= -S_5S_6 \\ U_{412} &= C_4U_{512}-S_4C_6 \\ U_{413} &= -C_4S_5 \\ U_{422} &= S_4U_{512}+C_4C_6 \\ U_{423} &= -S_4S_5 \\ U_{222} &= S_{23}U_{412}+C_{23}U_{522} \\ U_{223} &= S_{23}U_{413}+C_{23}C_5 \\ U_{224} &= C_{23}d_4+S_2a_2 \\ U_{212} &= C_{23}U_{412}-S_{23}U_{522} \\ U_{213} &= C_{23}U_{413}-S_{23}C_5 \\ U_{214} &= -S_{23}d_4+a_2C_2 \end{aligned} \tag{A.16}$$

followed by the evaluation of the elements of $\mathbf{T}_6$ given above. This computation corresponds to 6 sine/cosine pairs, 34 multiplies, and 17 additions.

The Jacobian

The Jacobian **J** matrix relates joint coordinate rates $\dot{\theta}_1$ through $\dot{\theta}_6$, to the Cartesian rates of the end of the manipulator **v**=$\dot{\mathbf{x}}$ by Eq. (2.13). The equivalent velocities in frame **T** due to the velocities at base **v** and **ω** are given by Eq. (2.11). A revolute joint, θ_i rotates about the $i-1$th. link **z** axis.

$$^{i-1}[0\,0\,0\,0\,1\,0]^T\dot{\theta}_i \tag{A.17}$$

Give **T** of the form in Eq(2.4), the relationship in Eq(2.11) becomes

$$^{i}[0\ \ C_{\alpha i}a_i\ \ -S_{\alpha i}a_i\ \ 0\ \ S_{\alpha i}\ \ C_{\alpha i}]^T\dot{\theta}_i \tag{A.18}$$

Using Eq(A.1),

$$\begin{array}{l} ^{1}[0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0]^T\dot{\theta}_1 \\ ^{2}[0\ \ a_2\ \ 0\ \ 0\ \ 0\ \ 1]^T\dot{\theta}_2 \\ ^{3}[0\ \ 0\ \ 0\ \ 0\ \ -1\ \ 0]^T\dot{\theta}_3 \\ ^{4}[0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0]^T\dot{\theta}_4 \\ ^{5}[0\ \ 0\ \ 0\ \ 0\ \ -1\ \ 0]^T\dot{\theta}_5 \\ ^{6}[0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1]^T\dot{\theta}_6 \end{array} \tag{A.19}$$

As joints two and three are parallel it is expected that use of the variable $\dot{\theta}_{23} = \dot{\theta}_2+\dot{\theta}_3$ might result in a simplification. Transforming the velocity due to $\dot{\theta}_2$ into link three, one obtains

$$^{3}[S_3a_2\ \ 0\ \ C_3a_2\ \ 0\ \ -1\ \ 0]^T\dot{\theta}_2 \tag{A.20}$$

Thus the relationship between rates $\dot{\theta}_2$ and $\dot{\theta}_3$ can replaced with the simplified expression between rates $\dot{\theta}_2$ and $\dot{\theta}_{23}$.

$$\begin{array}{l} ^{2}[0\ \ a_2\ \ 0\ \ 0\ \ 0\ \ 0]^T\dot{\theta}_2 \\ ^{3}[0\ \ 0\ \ 0\ \ 0\ \ -1\ \ 0]^T\dot{\theta}_{23} \end{array} \tag{A.21}$$

Each of these intermediate frame rates can now be transformed into link six rates using the differential transforms shown in Table 2A.

Table A2 Differential Transformations

Joint Variable	Initial Frame	Differential Transformation
1	1	$\mathbf{U}_2$
2	2	$\mathbf{U}_3$
23	3	$\mathbf{U}_4$
4	4	$\mathbf{U}_5$
5	5	$\mathbf{U}_6$
6	6	$\mathbf{I}$

Performing the transformations, one obtains the columns of the Jacobian

$$\begin{aligned}
&{}^6[U_{211}d_2+U_{421}U_{214}\ \ U_{212}d_2+U_{422}U_{214}\ \ U_{213}d_2+U_{423}U_{214}\ \ U_{221}\ \ U_{222}\ \ U_{223}]^T\dot{\theta}_1\\
&{}^6[U_{321}a_2\ \ U_{322}a_2\ \ U_{323}a_2\ \ 0\ \ 0\ \ 0]^T\dot{\theta}_2\\
&{}^6[-U_{411}d_4\ \ -U_{412}d_4\ \ -U_{413}d_4\ \ -U_{421}\ \ -U_{422}\ \ -U_{423}]^T\dot{\theta}_{23}\\
&{}^6[0\ \ 0\ \ 0\ \ U_{521}\ \ U_{522}\ \ C_5]^T\dot{\theta}_4\\
&{}^6[0\ \ 0\ \ 0\ \ -S_6\ \ -C_6\ \ 0]^T\dot{\theta}_5\\
&{}^6[0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1]^T\dot{\theta}_6
\end{aligned} \tag{A.22}$$

The Jacobian matrix has the following form:

$$\begin{bmatrix} \mathbf{J}_{11} & \mathbf{0} \\ \mathbf{J}_{21} & \mathbf{J}_{22} \end{bmatrix} \tag{A.23}$$

where:

$$\mathbf{J}_{11} = \begin{bmatrix} U_{211}d_2+U_{421}U_{214} & U_{321}a_2 & -U_{411}d_4 \\ U_{212}d_2+U_{422}U_{214} & U_{322}a_2 & -U_{412}d_4 \\ U_{213}d_2+U_{423}U_{214} & U_{323}a_2 & -U_{413}d_4 \end{bmatrix} \tag{A.24}$$

$$\mathbf{J}_{21} = \begin{bmatrix} U_{221} & 0 & -U_{421} \\ U_{222} & 0 & -U_{422} \\ U_{223} & 0 & -U_{423} \end{bmatrix} \tag{A.25}$$

$$\mathbf{J}_{22} = \begin{bmatrix} U_{521} & -S_6 & 0 \\ U_{522} & -C_6 & 0 \\ C_5 & 0 & 1 \end{bmatrix} \tag{A.26}$$

Inverse Jacobian

The Inverse Jacobian is given by:

$$\mathbf{J}_{11}^{-1} = \begin{bmatrix} \mathbf{J}_{11}^{-1} & \mathbf{0} \\ -\mathbf{J}_{22}^{-1}\mathbf{J}_{21}\mathbf{J}_{11}^{-1} & \mathbf{J}_{22}^{-1} \end{bmatrix} \tag{A.27}$$

and

$$\mathbf{J}_{22}^{-1} = \frac{-1}{S_5} \begin{bmatrix} -C_6 & S_6 & 0 \\ -U_{522} & U_{521} & 0 \\ U_{511} & U_{512} & -S_5 \end{bmatrix} \tag{A.28}$$

$\mathbf{J}_{11}$ is too complicated to invert symbolically. However, considering the three columns of $\mathbf{J}_{11}$ as vectors **a**, **b** and **c**:

$$\mathbf{J}_{11} = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}] \tag{A.29}$$

Then its inverse is:

$$\mathbf{J}_{11}^{-1} = \frac{1}{\mathbf{a} \cdot \mathbf{b} \times \mathbf{c}} \begin{bmatrix} \mathbf{b} \times \mathbf{c} \\ \mathbf{c} \times \mathbf{a} \\ \mathbf{a} \times \mathbf{b} \end{bmatrix} \tag{A.30}$$

Inverse Kinematics

In order to obtain the solution, Eq. (2.9) is used. Vs are obtained as follows. Premultiplying $\mathbf{T}_6$ by A_1^{-1}, one obtains $\mathbf{V}_{1j}$

$$\mathbf{V}_{1j} = \begin{bmatrix} C_1 x + S_1 y \\ z \\ S_1 x - C_1 y \\ w \end{bmatrix} \tag{A.31}$$

Define

$$\begin{aligned} V_{11} &= C_1x+S_1y \\ V_{13} &= S_1x-C_1y \end{aligned} \tag{A.32}$$

Then premultiplying $\mathbf{V}_{1j}$ by A_2^{-1}, one obtains $\mathbf{V}_{2j}$

$$\mathbf{V}_{2j} = \begin{bmatrix} V_{21} = C_2V_{11}+S_2z-a_2w \\ V_{22} = -S_2V_{11}+C_2z \\ V_{23} = V_{13}-d_2w \\ w \end{bmatrix} \tag{A.33}$$

Premultiplying $\mathbf{V}_1$ by $\mathbf{A}_{23}^{-1}$, one obtains $\mathbf{V}_3$

$$\mathbf{V}_{3j} = \begin{bmatrix} V_{31} = C_{23}V_{11}+S_{23}z-a_2C_3w \\ V_{32} = -V_{13}+d_2w \\ V_{33} = -S_{23}V_{11}+C_{23}z+a_2S_3w \\ w \end{bmatrix} \tag{A.34}$$

Premultiplying by A_4^{-1}, one obtains $\mathbf{V}_{4j}$

$$\mathbf{V}_{4j} = \begin{bmatrix} V_{41} = C_4V_{31}+S_4V_{32} \\ V_{42} = V_{33}-d_4w \\ V_{43} = S_4V_{31}-C_4V_{32} \\ w \end{bmatrix} \tag{A.35}$$

Finally premultiplying by $\mathbf{A}_5^{-1}$, one obtains $\mathbf{V}_{5j}$

$$\mathbf{V}_{5j} = \begin{bmatrix} V_{51} = C_5V_{41}+S_5V_{42} \\ -V_{43} \\ V_{53} = -S_5V_{41}+C_5V_{42} \\ w \end{bmatrix} \tag{A.36}$$

The solution is now obtained by solving the six equations in Eq(2.9). First equate $\mathbf{V}_0$ to $\mathbf{U}_1$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ? & C_1U_{212}-S_1U_{422} & C_1U_{213}-S_1U_{423} & C_1U_{214}+S_1d_3 \\ ? & S_1U_{212}+C_1U_{422} & S_1U_{213}+C_1U_{423} & S_1U_{214}-C_1d_3 \\ ? & U_{222} & U_{223} & U_{224} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.37}$$

Equating the 14 and 24 elements leads to

$$\begin{aligned} p_x &= C_1U_{214}+S_1d_2 \\ p_y &= S_1U_{214}-C_1d_2 \end{aligned} \tag{A.38}$$

Eliminating U_{214}, one gets

$$S_1p_x-C_1p_y = d_2 \tag{A.39}$$

Let:

$$r\sin_\phi = p_y \quad r\cos_\phi = p_x \quad \text{with } r>0$$

Then

$$r = +\sqrt{p_x^2+p_y^2} \quad \text{and} \quad \phi = \tan^{-1}\frac{p_y}{p_x}$$

One obtains

$$\sin(\theta_1-\phi) = \frac{d_2}{r} \tag{A.40}$$

This equation has two solutions corresponding to the *left* and *right* configurations. The one corresponding to *left* is

$$\theta_1 = \tan^{-1}\frac{p_y}{p_x}+\sin^{-1}\frac{d_2}{r} \tag{A.41a}$$

The solution corresponding to *right* is

$$\theta_1 = \tan^{-1}\frac{p_y}{p_x}+\pi-\sin^{-1}\frac{d_2}{r} \tag{A.41b}$$

Next equating $\mathbf{V}_1$ to $\mathbf{U}_2$, one obtains

$$\begin{bmatrix} C_1x+S_1y \\ z \\ S_1x-C_1y \\ w \end{bmatrix} = \begin{bmatrix} ? & U_{212} & U_{213} & -S_{23}d_4+a_2C_2 \\ ? & U_{222} & U_{232} & C_{23}d_4+a_2S_2 \\ ? & -U_{422} & -U_{423} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.42}$$

Define

$$\begin{aligned} V_{114} &= -S_{23}d_4+a_2C_2 \\ V_{124} &= p_z = C_{23}d_4+a_2S_2 \end{aligned} \tag{A.43}$$

Isolating the 23 terms, one gets

$$\begin{aligned} V_{114}-a_2C_2 &= -S_{23}d_4 \\ p_z-a_2S_2 &= C_{23}d_4 \end{aligned} \tag{A.44}$$

then squaring and adding, one obtains

$$C_2V_{114}+S_2p_z = \frac{a_2^2-d_4^2+V_{114}^2+p_z^2}{2a_2} \tag{A.45}$$

letting

$$r\sin\phi = p_z \quad r\cos\phi = V_{114} \quad \text{with } r>0$$

Then

$$r = +\sqrt{V_{114}^2+p_z^2} \quad \text{and} \quad \phi = \tan^{-1}\frac{p_z}{V_{114}}$$

and making the substitution one obtains

$$\cos(\theta_2-\phi) = \frac{a_2^2-d_4^2+V_{114}^2+p_z^2}{2a_2r} \tag{A.46}$$

This equation again has two solutions corresponding to *elbow up* and *elbow down.* Defining

$$\psi = \cos^{-1}\frac{a_2^2-d_4^2+V_{114}^2+p_z^2}{2a_2r} \tag{A.47}$$

then the *elbow down* solution is

$$\theta_2 = \tan^{-1}\frac{p_z}{V_{114}}+\psi \tag{A.48a}$$

A second solution corresponds to *elbow up*

$$\theta_2 = \tan^{-1}\frac{p_z}{V_{114}}-\psi \tag{A.48b}$$

θ_3 is then obtained from the next equation $\mathbf{V}_4 = \mathbf{U}_3$

$$\begin{bmatrix} C_2V_{11}+S_2z-a_2w \\ -S_2V_{11}+C_2z \\ V_{13}-d_2w \\ w \end{bmatrix} = \begin{bmatrix} ? & U_{312} & U_{313} & -S_3d_4 \\ ? & U_{322} & U_{323} & C_3d_4 \\ ? & -U_{422} & -U_{423} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.49}$$

from which one obtains

$$\begin{aligned} V_{214} &= -S_3d_4 \\ V_{224} &= C_3d_4 \end{aligned} \tag{A.50}$$

$$\tan\theta_3 = \frac{-V_{214}}{V_{224}} \tag{A.51}$$

and

$$\theta_3 = \tan^{-1}\frac{a_2-C_2V_{114}-S_2p_z}{C_2p_z-S_2V_{114}} \tag{A.52}$$

θ_4 is obtained from

$$\mathbf{V}_3 = \mathbf{U}_4 \tag{A.53}$$

or

$$\begin{bmatrix} V_{31} \\ V_{32} \\ V_{33} \\ w \end{bmatrix} = \begin{bmatrix} ? & U_{412} & -C_4S_5 & 0 \\ ? & U_{422} & -S_4S_5 & 0 \\ ? & U_{522} & C_5 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.54}$$

Expressions for the sine and cosine of θ_4 can be obtained as

$$S_4 = -\frac{1}{S_5} V_{323} \qquad C_4 = -\frac{1}{S_5} V_{313} \tag{A.55}$$

The sign of S_5 determines the third configuration, *flip* and *non-flip*. When *flip*ped,

$$\theta_4 = \tan^{-1} \frac{V_{323}}{V_{313}} \tag{A.56a}$$

Otherwise,

$$\theta_4 = \tan^{-1} \frac{-V_{323}}{-V_{313}} \tag{A.56b}$$

θ_4 is undefined if $S_5 = 0$ as the manipulator runs into a singular point. V_{323} and V_{313} can be evaluated with their recursive definition to obtain:

$$\begin{aligned} V_{323} &= C_1 a_y - S_1 a_x \\ V_{113} &= C_1 a_x + S_1 a_y \\ V_{313} &= C_{23} V_{113} + S_{23} a_z \end{aligned} \tag{A.57}$$

From the next equation

$$\mathbf{V}_4 = \mathbf{U}_5$$

one obtains

$$\begin{bmatrix} C_4 V_{31} + S_4 V_{32} \\ V_{33} - d_4 w \\ S_4 V_{31} - C_4 V_{32} \\ w \end{bmatrix} = \begin{bmatrix} ? & U_{512} & -S_5 & 0 \\ ? & U_{522} & C_5 & 0 \\ ? & -C_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.58}$$

One obtains directly that

$$\begin{aligned} S_5 &= -C_4 V_{313} - S_4 V_{323} \\ C_5 &= -S_{23} V_{113} + C_{23} a_z \end{aligned} \tag{A.59}$$

where V_{313} and V_{323} are defined above. Therefore,

$$\theta_5 = \tan^{-1} \frac{S_5}{C_5} \tag{A.60}$$

Finally equating

$$\begin{bmatrix} C_5 V_{41} + S_5 V_{42} \\ -V_{43} \\ -S_5 V_{41} + C_5 V_{42} \\ w \end{bmatrix} = \begin{bmatrix} ? & -S_6 & 0 & 0 \\ ? & C_6 & 0 & 0 \\ ? & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.61}$$

Equations for S_6 and C_6 are obtained directly by equating the first two elements of the second column.

$$\begin{aligned} S_6 &= -C_5 V_{412} - S_5 V_{422} \\ C_6 &= -V_{432} \end{aligned} \tag{A.62}$$

The expressions on the right hand side are evaluated as

$$\begin{aligned} V_{412} &= C_4 V_{312} - S_4 V_{132} \\ V_{422} &= V_{332} \\ V_{432} &= S_4 V_{312} + C_4 V_{132} \\ V_{312} &= C_{23} V_{112} + S_{23} o_z \\ V_{332} &= -S_{23} V_{112} + C_{23} o_z \\ V_{132} &= S_1 o_x - C_1 o_y \\ V_{112} &= C_1 o_x + S_1 o_y \end{aligned} \tag{A.63}$$

θ_6 is finally obtained by

$$\theta_6 = \tan^{-1} \frac{S_6}{C_6} \tag{A.64}$$

Appendix B

Dynamics Equations of PUMA 560 Manipulator

The constants in the gravity and inertia coefficients for PUMA 560 are listed in this appendix. The derivation of the equations can be found in [Paul, Ma, and Zhang 1983]. Radii of gyration for the PUMA 560 manipulator are given in Table A3.

Table A3. Radii of Gyration for the PUMA 560

link	$k_{xx}^2 (cm^2)$	$k_{yy}^2 (cm^2)$	$k_{zz}^2 (cm^2)$
1	451	451	58
2	566	1847	1408
3	673	679	36
4	326	21	32
5	7	10	7
6	34	34	1

The centers of mass of each link are the following

$$ {}^6\bar{r}_6 = \begin{bmatrix} 0 \\ 0 \\ \bar{z}_6 \\ 1 \end{bmatrix} \quad {}^5\bar{r}_5 = \begin{bmatrix} 0 \\ 0 \\ \bar{z}_5 \\ 1 \end{bmatrix} \quad {}^4\bar{r}_4 = \begin{bmatrix} 0 \\ \bar{y}_4 \\ 0 \\ 1 \end{bmatrix} \tag{B.1} $$

$$ {}^3\bar{r}_3 = \begin{bmatrix} 0 \\ 0 \\ \bar{z}_3 \\ 1 \end{bmatrix} \quad {}^2\bar{r}_2 = \begin{bmatrix} \bar{x}_2 \\ 0 \\ \bar{z}_2 \\ 1 \end{bmatrix} \quad {}^1\bar{r}_1 = \begin{bmatrix} 0 \\ 0 \\ \bar{z}_1 \\ 1 \end{bmatrix} \tag{B.2} $$

There are four constants in gravity coefficients as indicated by Eq. (4.9). Their symbolic expressions are:

$$c^{l}_{50} = -g\,(m_6\bar{z}_6 + m_5\bar{z}_5) \tag{B.3}$$

$$c^{l}_{31} = -(m_3 + m_4 + m_5 + m_6)\,g a_3 \tag{B.4}$$

$$c^{l}_{32} = -(m_3\bar{z}_3 + m_4 d_4 - m_4\bar{y}_4 + m_5 d_4 + m_6 d_4)\,g \tag{B.5}$$

$$c^{l}_{21} = -\left[m_2\bar{x}_2 + (m_3 + m_4 + m_5 + m_6)\,a_2\right] g \tag{B.6}$$

A total of 15 constants exist to describe the effective inertia coefficients. For joint six through three,

$$b^{l}_{60} = m_6 k^{2}_{6zz} \tag{B.7}$$

$$b^{l}_{50} = m_5 k^{2}_{5yy} + m_6 k^{2}_{6xx} \tag{B.8}$$

$$b_{40} = m_4 k^{2}_{4yy} + m_5 k^{2}_{5xx} \tag{B.9}$$

$$\begin{aligned} b^{l}_{30} = {} & m_3(k^{2}_{3yy} + a^{2}_{3}) + m_4(k^{2}_{4xx} + d^{2}_{4} + a^{2}_{3} - 2d_4\bar{y}_4) + \\ & m_5(k^{2}_{5xx} + d^{2}_{4} + a^{2}_{3}) + m_6(d^{2}_{4} + d^{2}_{3}) \end{aligned} \tag{B.10}$$

$$b^{l}_{31} = 2m_5\bar{z}_5 d_4 + 2m_6\bar{z}_6 d_4 \tag{B.11}$$

For joint two,

$$\begin{aligned} b^{l}_{20} = {} & m_2 k^{2}_{2zz} + m_3 k^{2}_{3yy} + m_4 k^{2}_{4xx} + m_5 k^{2}_{5xx} + m_2\,(a^{2}_{2} + 2\bar{x}_2 a_2) \\ & + m_3 a^{2}_{3} + (m_4 + m_5 + m_6)(d^{2}_{4} + a^{2}_{3}) - 2m_4\bar{y}_4 d_4 \end{aligned} \tag{B.12}$$

$$b^{l}_{25} = 2a_2\left[m_3\bar{z}_3 + d_4\,(m_4 + m_5 + m_6)\right] \tag{B.13}$$

$$b_{26} = 2\bar{z}_5 a_2 + 2\bar{z}_6 a_2 \tag{B.14}$$

For joint one,

$$b^{l}_{10} = m_1 k^2_{1yy} + m_2 k^2_{2xx} + m_3 k^2_{3xx} + m_4 k^2_{4xx} + m_5 k^2_{5xx} +$$

$$(m_3 + m_4 + m_5 + m_6) d^2_3 + (m_4 + m_5 + m_6) d^2_4 - 2 m_4 \overline{y}_4 d_4 \qquad \text{(B.15)}$$

$$b^{l}_{11} = (m_2 + m_3 + m_4 + m_5 + m_6) a_2^2 + 2 m_2 \overline{x}_2 a_2 + m_{2(} k^2_{2yy} - k^2_{2xx}) \qquad \text{(B.16)}$$

$$b^{l}_{12} = (m_3 + m_4 + m_5 + m_6) a^2_3 - (m_4 + m_5 + m_6) d^2_4 + 2 m_4 \overline{y}_4 d_4 +$$

$$m_3 (k^2_{3zz} - k^2_{3xx}) \qquad \text{(B.17)}$$

$$b^{l}_{13} = 2(m_3 + m_4 + m_5 + m_6) a_3 a_2 \qquad \text{(B.18)}$$

$$b^{l}_{14} = 2a_2 (m_3 \overline{z}_3 + (m_4 + m_5 + m_6) d_4 - m_4 \overline{y}) \qquad \text{(B.19)}$$

$$b^{l}_{15} = 2a_3 (m_3 \overline{z}_3 + (m_4 + m_5 + m_6) d_4 - m_4 \overline{y}) \qquad \text{(B.20)}$$

The coupling inertia coefficient D_{23} is defined by

$$b^{l}_{230} = m_3 k^2_{3yy} + m_4 k^2_{4xx} + m_5 k^2_{5xx} + m_6 k^2_{6xx} + (m_3 + m_4 + m_5 + m_6) a^2_3$$

$$+ (m_4 + m_5 + m_6) d^2_4 + (2 m_4 \overline{y}_4 + m_6 \overline{z}_6) d_4 \qquad \text{(B.21)}$$

$$b^{l}_{231} = (m_3 + m_4 + m_5 + m_6) a_2 a_3 \qquad \text{(B.22)}$$

$$b^{l}_{232} = m_3 \overline{z}_3 a_2 + (m_4 + m_5 + m_6) a_2 d_4 + \overline{y}_4 a_2 m_4 \qquad \text{(B.23)}$$

Appendix C

Distributed Trajectory Generator and Hybrid Controller

Supervisor

The supervisor figures out the current T_6 and sends the necessary information to the joint processors. During a transition, it computes the *r* using the Eq. (3.15).

```
/*
 * setp.c - contains set point generation functions
 */

/*
 * setp.c - set point oenerater
 *        This program is constructed as a finite state machine based
 *        on "motion trajectory spec and generation" (2nd ISRR).
 *        State zero scorreponds to the idle state when there is not
 *        a new motion request fetched.  State one corresponds to
 *        the straight line segment of the motion.  State two corres-
 *        ponds to one sample period before the transition and there is
 *        a next motion request to have been dequeued.  State
 *        three corresponds to the beginning of the transition.
 *        State four corresponds to the transition period.  State
 *        five is a intermediate state when the arm is brought to
 *        rest because there is no next entry in the motion queue
 *        and the arm is in the middle of the motion.
 */

# include "../h/datdef.h"
# include "../h/condef.h"
# include "../h/fundef.h"

# define SEGT_DEF 10.0           /* default segment and acceleration time */
# define ACCT_DEF 0.3         /* in seconds */
# define SAMPLE   0.005  /* sample period in seconds */
# define NOP    200             /* number of points SEG/SAMPLE*/

/*
```

```
 * setpoint() - generates the next set point
 */

double send();
extern PST_PTR dstpst;

int setpoint()
{
     static PST_PTR newpst;
     static float   Time = SEGT_DEF,    /* clock */
                    Tseg = SEGT_DEF,    /* segment time */
                    Tacc = ACCT_DEF;    /* acceleration time */
     static int     state = STATE0,     /* of what to compute */
                    mode = JNTM,   /* mode of the motion */
                    NM = NO;  /* a boolean for New Motion */

     static REQ     *newmove; /* new motion request */

     static NOAP    exp1, exp2,
                    *expr1 = &exp1,     /* 1st of the two possible T6's */
                    *expr2 = &exp2;     /* 2nd of ... */

     REQ       *move;         /* motion request var */
     float     r,
               h,             /* segment parameter */
               p;

     /* write appropriate values to the joint processors */

     switch(state) {
     case STATE0:   /* idle case */
          /* "s_expl" sends the first T6 to joint processor */
          /* "getEX" solves for T6 from the position equation */
          s_expl(getEX(expr1, dstpst, 1.0), mode, 0.0, state);

          /* "dequeue" looks into the motion request queue */
          if((move = dequeue()) != NULL) {
               newmove = move;
               state = STATE3;
               newpst = newmove->pst;
               NM = YES;
               if(newmove->mot->mode == JNTM) {
                    /* sends the second possible T6 to joints */
                    s_exp2(getEX(expr2, newpst, 0.0));
               } else {
                    /* initializes the drive transform */
                    initD(expr1, newpst);
                    s_exp2(getLDR(expr2, newpst, 0.0));
               }
               send_status(STM);
          }
          break;
```

```
case STATE1:   /* straight line segment */
     h = Time/Tseg;
     r = 1.0 - h;
     if(mode == JNTM) {
          expr1 = getEX(expr1, dstpst, h);
     } else {
          expr1 = getLDR(expr1, dstpst, r);
     }
     s_expl(expr1, mode, r, state);
     if(!NM) { /* keep look if there is no next motion yet */
          if((move = dequeue()) != NULL) {
              newmove = move;
              NM = YES;
              Tacc = newmove->mot->tacc;
              if(Time > Tseg-Tacc-SAMPLE) { /* do have a move */
               state = STATE2;          /* but a bit late */
               break;
              }
          }
     }
     Time += SAMPLE;
     if(Time >= Tseg - Tacc - SAMPLE) {
          if(!NM) {      /* come to a stop */
               newmove = newmove;
          }
          state = STATE2;
     }
     break;
case STATE2:   /* one sample period before the transition */
     h = Time/Tseg;
     r = 1.0 - h;
     if(mode == JNTM) {
          expr1 = getEX(expr1, dstpst, h);
     } else {
          expr1 = getLDR(expr1, dstpst, r);
     }
     s_expl(expr1, mode, r, state);
     newpst = newmove->pst;   /* = dstpst if !NM */
     s_exp2(getEX(expr2, newpst, 0.));
        state = STATE3;
     break;
case STATE3:   /* beginning of the transition */
     expr1 = getEX(expr1, dstpst, 1.0);
     if(NM) {
          Tseg = newmove->mot->tseg;
          mode = newmove->mot->mode;
          Tacc = newmove->mot->tacc;
          NM = NO;
     } else {
          Tseg = Tacc;   /* come to a stop */
     }
     getEX(expr1, dstpst, 1.0);
```

```
          initD(expr1, newpst);
          dstpst = newpst;
          Time = -Tacc + SAMPLE;
          s_exp1(expr1, mode, r, state);     /* r undefined */
          s_exp2(getEX(expr2, newpst, 0.0));
          s_tacc(Tacc);
          state = STATE4;
          break;
     case STATE4:   /* during the transition */
          p = (Time + Tacc)/(2.0*Tacc);
          h = ((2.*p-6.)*p+5.0)*p*p*p*p*Tacc/Tseg;
          r = 1.0 - h;
          if(mode == JNTM) {
               expr1 = getEX(expr1, dstpst, h);
          } else {
               expr1 = getLDR(expr1, dstpst, r);
          }
          s_exp1(expr1, mode, r, state);
          send_p(p);
          Time += SAMPLE;
          if(Time > Tacc)
               state = (Tacc >= Tseg)? STATE0:STATE1;
          break;
     default:
          printf("unknown state: %d0, state);
          return;
     }
     send();          /* This really sends. */
     return(1);       /* for debugging only */
}
```

Joint Process

Three functions corresponding to three stages of the joint process exist on each joint processor. The first computes the joint set-point without compliance specification based on the T_6 received from the supervisor. The second computes the real joint set-point with compliance consideration and then the joint accelerations for dynamics compensation. The third function computes joint torques based on its joint position error and all the joint accelerations. In the following, the three functions for joint one are given. Other joints have similar functions with differing joint parameters such as inertias and controller gains.

```
/*
 * j1.c - The first part of the joint set point process;
 *     interrupt driven from the supervisor;
 *        communicates with the supervisor via the mail structure
 *     This segment of the program is identical for all joints.
 *     Refer to "Motion Trajectory Generation" for details.
 */

# include "../h/datdef.h"
# include "../h/condef.h"
# include "../h/comm.h"
# define SAMPLE          0.005     /* sampling time */
# define ONEBYTAU    (1.0/SAMPLE)

/*
 * jsetp() - computes joint error without considering compliance
 */

extern S_MAIL MAIL; /* Structure sent by the supervisor */
J_MAIL    JMAIL;          /* Things to be sent back to supervisor */

int jsetp()
{
     static float   thBC = 0.0,
                    thold = 0.0,
                    thdot = 0.0,
                    thtmp = 0.0;

     static float a0, a1, a4, a5, a6, a7;

     double solve();
     float     p, theta, thchg, thB, thC;

     switch(MAIL.state) {
     case STATE0:
          theta = solve(MAIL.expr1);
          thdot = (theta - thold)*ONEBYTAU;
          if(MAIL.status == STM) {
               MAIL.status = NT;
               thtmp = solve(MAIL.expr2);
          }
          thold = theta;
          break;
     case STATE1:
          if(MAIL.mode == JNTM) {
               thchg = solve(MAIL.expr1);
               theta = MAIL.r*thBC + thchg;
          } else {
               theta = solve(MAIL.expr1);
          }
          break;
     case STATE2:
          if(MAIL.mode == JNTM) {
               thchg = solve(MAIL.expr1);
               theta = MAIL.r*thBC + thchg;
          } else {
               theta = solve(MAIL.expr1);
          }
          thtmp = solve(MAIL.expr2);
          thdot = (theta - thold)*ONEBYTAU;
          break;
     case STATE3:
          theta = thold + thdot*SAMPLE;
          thB = solve(MAIL.expr1);
          thC = solve(MAIL.expr2);
          a0 = theta - thB;
          thBC = thB - thC;
          a1 = 2.*MAIL.tacc*(thdot - (thC - thtmp)*ONEBYTAU);
          a4 = -20.*a1 - 35.*a0;
          a5 = 45.*a1 + 84.*a0;
          a6 = -36.*a1 - 70.*a0;
          a7 = 10.*a1 + 20.*a0;
          break;
     case STATE4:
          p = MAIL.p;
          theta = ((((a7*p + a6)*p + a5)*p + a4)*p*p*p + a1)*p + a0;
          if(MAIL.mode == JNTM) {
               theta += solve(MAIL.expr1);
               theta += MAIL.r*thBC;
          } else {
               theta += solve(MAIL.expr1);
          }
          thdot = (theta - thold)*ONEBYTAU;
          break;
     default:
          break;
     }
     JMAIL.theta.th1 = theta;
     thold = theta;
}

/*
 * solve() - inverse kinematics for one joint (j1 of PUMA 260 in this program).
 */

double atan2(),
       asin(),
       sqrt();

double solve(trsf)
NOAP *trsf;
{
     float x   = trsf->p.x, y = trsf->p.y;
     float u, v;
```

```
    if(c_left)
         return (atan2(y, x) + asin(D2/sqrt(x*x+y*y));
    else
         return(PI + atan2(y, x) - asin(D2/sqrt(x*x + y*y)));
}
```

Joint Compliance Process

```
# include <math.h>
# include "../../h/datdef.h"
# include "../../h/comm.h"
# include "../../h/condef.h"
/*
 * comply() - computes joint error based on compliance consideration
 *         (written for joint one).
 */

extern PARCEL dyn_cmy;
extern J_MAIL MAIL;

double comply(err)
JNT_PTR err;
{
     JNT_PTR cmymat;
     float error;

     cmymat = &dyn_cmy.Crow;
     if(MAIL.comply & JNT1) {
          error = err->th1*cmymat->th1 + err->th2*cmymat->th2 +
                  err->th23*cmymat->th23 + err->th4*cmymat->th4 +
                  err->th5*cmymat->th5 + err->th6*cmymat->th6;
     } else {
          error = err->th1;
     }
     return(error);
}
```

Joint Torque Computation

```
# include "../../h/datdef.h"
# include "../../h/comm.h"
# define KP 1.0
# define KI 1.0          /* to be adjusted by experiments */
# define KD 1.0
# define SAMPLE          0.005
# define SAMPLE2     (1/(SAMPLE*SAMPLE)) /* scaling constant */

/*
 * dyn() - computes torques and drives the joint
 */

extern PARCEL dyn_cmy;

double dyn(err, thacc)
JNT_PTR err,
     thacc;
{
     static float   errold, errint;
     JNT_PTR   inertia;
     float     errdot, torque;

     errdot = err->th1 - errold;   /* error derivative * SAMPLE */
     errint += err->th1;      /* error integration / SAMPLE */
     inertia = &dyn_cmy.inertia;

     /* inertial effect */
     torque = (inertia->th1*thacc->th1 + inertia->th2*thacc->th2
           + inertia->th3*thacc->th23 + inertia->th4*thacc->th4
           + inertia->th5*thacc->th5 +
           inertia->th6*thacc->th6)*SAMPLE2;

     /* PID + gravity */
     torque = KP*err->th1 + KD*errdot + KI*errint +
           inertia->th1*thacc->th1 + inertia->th2*thacc->th2
           + inertia->th3*thacc->th23 + inertia->th4*thacc->th4
           + inertia->th5*thacc->th5 + inertia->th6*thacc->th6
           + dyn_cmy.gravity;
     return(torque);
}
```