

**IMPLEMENTATION OF ROBOTIC
FORCE CONTROL WITH
POSITION ACCOMMODATION**

NAGW-1333

by

Michael J. Ryan

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

June 1992

CIRSSE REPORT #118



© Copyright 1992

by

Michael J. Ryan

All Rights Reserved

CONTENTS

LIST OF TABLES	v
LIST OF FIGURES	vi
ACKNOWLEDGMENT	viii
ABSTRACT	ix
1. INTRODUCTION	1
1.1 Goals of the Project	2
1.2 Brief Historical Review	3
1.3 Report Organization	3
2. THEORY AND ANALYSIS	5
2.1 Position-Based Force Control	5
2.1.1 Analysis of Force Mechanisms	5
2.2 Position Accommodation	7
2.2.1 The Phi-Matrix	11
2.3 Conclusion	14
3. TESTBED DESCRIPTION	15
3.1 HARDWARE	15
3.1.1 6 DOF PUMA Robots	17
3.1.2 3 DOF Transporter Platforms	19
3.1.3 Computer-Control System	20
3.1.4 Force Sensors	21
3.1.5 Pneumatic Grippers	22
3.2 SOFTWARE	22
3.2.1 CIRSSE Testbed Operating System (CTOS)	23
3.2.2 Motion Control System (MCS)	24
3.3 Conclusion	26

4. FORCE CONTROL IMPLEMENTATION	27
4.1 Trajectory Generation	27
4.1.1 Integration of Position Accommodation	27
4.2 Position Accommodation Function	30
4.2.1 Tool to Sensor Transform	31
4.2.2 Provision for F/T Sensor Rotations	32
4.2.3 Biasing of the F/T Sensor	33
4.3 Dual Arm Implementation	34
4.4 Conclusion	35
5. EXPERIMENTS AND RESULTS	36
5.1 Free-Air Tests	36
5.2 Contact Tests	39
5.2.1 Insertion Tests	39
5.3 Two-Arm Tests	43
5.4 Conclusion	45
6. DISCUSSION AND CONCLUSIONS	47
6.1 Position-Based Force Control	47
6.2 Performance of the PAC Force Control	50
6.2.1 Force-Filtering	50
6.2.2 Slow-Motions	51
6.2.3 Implementation of Compliant Rotations	51
6.2.4 Limitations of PAC Force Control	53
6.2.5 Dual-Arm Manipulation	54
6.3 Future Work	55
6.4 Conclusion	56
LITERATURE CITED	58
APPENDICES	61
A. SOFTWARE SOURCE CODE	61

LIST OF TABLES

Table 3.1	Testbed Joint Parameters	17
Table 3.2	Rated Capabilities of PUMA 560 Arm	18
Table 3.3	Modules used in VMEbus Cage	20
Table 3.4	Force/Torque Sensor System	21
Table 4.1	Cycle-Periods for Motion Control Tasks	29
Table 5.1	Insertion Impedance: PD Position-Controller	42
Table 5.2	Insertion Impedance: PID Position-Controller	42
Table 6.1	Approximate Stiffness of Force Mechanisms: Z-axis	48
Table 6.2	Approximate Force Resolution: Z-axis	48

LIST OF FIGURES

Figure 2.1	Position-Based Force Control	6
Figure 2.2	(a) Ideal Arm and Spring; (b) Force Function Block	6
Figure 2.3	Mass-Spring-Damper Impedance Models: (a) Linear; (b) Ro- tational	8
Figure 2.4	(a) Mounting of Force/Torque Sensor; (b) Kinematic Frames of Flange, Sensor, and Tool-Tip	12
Figure 3.1	CIRSSE Robotic Testbed	16
Figure 3.2	CIRSSE Testbed Computer System	19
Figure 3.3	Architecture of MCS under CTOS	25
Figure 4.1	Trajectory Generator Data Flow	28
Figure 4.2	Trajectory Generator with Position Accommodation	28
Figure 4.3	Position Accommodation: Continuous Model	30
Figure 4.4	Position Accommodation: Discrete Model	32
Figure 4.5	Dual Independent Trajectory Generators	35
Figure 5.1	Manipulator in "Safe" Position	37
Figure 5.2	Linear Motion in Free-Air (Test1a)	37
Figure 5.3	Linear Motion in Free-Air (Test1b)	38
Figure 5.4	Impact with Box: PD Position-Control (Test2a)	40
Figure 5.5	Impact with Box: PD Position-Control (Test2b)	40
Figure 5.6	Impact with Box: PID Position-Control (Test2c)	41
Figure 5.7	Impact with Box: PID Position-Control (Test2d)	41
Figure 5.8	Two-Arm Test: No Bias-Force (Test3a)	44
Figure 5.9	Two-Arm Test: Compression (Test3b)	44
Figure 5.10	Two-Arm Test: Tension (Test3c)	45

Figure 6.1 Dual Cooperative Trajectory Generators 56

ACKNOWLEDGMENT

I would like to express my heartfelt thanks to my advisor Steve Murphy. His guidance, understanding, and patience throughout this project was without end. I must also thank Alan Desrochers, without whose support I could not have completed this project. Special thanks goes out to Lee Wilfinger, Kevin Holt, and the rest of the CIRSSSE crew I had the pleasure of working with.

I am forever grateful to my Mom and Dad for their continued moral and financial support. Lastly, I would like to thank my friends Ann and April for their encouragement and belief in me.

ABSTRACT

As the need for robotic manipulation in fields such as manufacturing and telerobotics increases, so does the need for effective methods of controlling the interaction forces between the manipulators and their environment. Position Accommodation is a form of robotic force control where the nominal path of the manipulator is modified in response to forces and torques sensed at the tool-tip of the manipulator. The response is tailored such that the manipulator emulates a mechanical impedance to its environment. Position Accommodation falls under the category of position-based robotic force control, and may be viewed as a form of Impedance Control.

This project explores the practical implementations of Position Accommodation into an 18 degree-of-freedom robotic testbed consisting of two PUMA 560 arms mounted on two 3 DOF positioning platforms. Single and dual-arm architectures for Position Accommodation are presented along with some experimental results. Characteristics of position-based force control are discussed, along with some of the limitations of Position Accommodation.

CHAPTER 1

INTRODUCTION

It has long been a goal in robotic development that manipulators take on the complex assembly and manufacturing tasks normally performed by human laborers. For these tasks, the interaction forces between the manipulator and its environment of workpieces, fixtures, obstacles, etc., can be significant, and must be controlled to prevent deformation or breakage. For any assembly task with moderately tight tolerances, it will be necessary for the manipulator to “comply” to unforeseen forces due to misalignment errors, incomplete models, etc. A classic example is the task of inserting a tapered peg into a hole. Other assembly and manufacturing tasks require the application of a desired force, such as in crimping, drilling, and grinding. For most of these tasks, some form of active force-feedback control will be necessary to achieve consistent, reliable performance.

Position Accommodation is a form of robotic force control where the nominal path of the manipulator is modified in response to forces and torques sensed at the *tool-tip* of the manipulator. The response can be controlled such that the manipulator will appear as a mechanical *impedance* to its environment, with compliance capabilities in all six degrees-of-freedom (DOF). Position Accommodation (PAC) uses the position-control system of the robotic arm directly so it can be implemented on many robotic arms without modification of their existing controllers. Through an impedance specification, the manipulator’s response to external forces can be tailored to suit the desired assembly task. In addition, bias-forces along any of the six degrees of compliance can be specified, so that insertion forces/torques can be applied.

Multiple-Manipulators Robotic manipulation with multiple arms has shown promise for significantly exceeding the capabilities of a single manipulator. Manipulation of massive payloads, handling of tools, complex assembly tasks, extensions to workspace, etc., are some of the advantages foreseen for multi-arm systems. To enable any form of multi-arm manipulation the arms must again have the ability to “comply” to unforeseen forces due to misalignment, incomplete models, etc. Multiple-arms also have the capability for exerting *internal* forces and torques on their workpiece that single arms cannot, and for this some form of active force control will be needed. Position Accommodation is seen as a promising method for multi-arm force control.

1.1 Goals of the Project

The central goal of this project was the implementation of Position Accommodation force control onto the 18 DOF robotic testbed developed here at the Center for Intelligent Robotics Systems For Space Exploration (CIRSSE). The testbed consists of two 6 DOF PUMA 560 arms mounted onto two 3 DOF positioning platforms. The testbed was developed for research into space-based robotic applications, where the areas of motion control, trajectory generation, task planning & coordination, vision, etc., could be explored and developed. One particular application looked at is the assembly of the struts and nodes comprising the truss structure of a space-station.

The PAC software developed for this project will become part of the installed library of functions available for researchers wanting to perform assembly experiments on the CIRSSE testbed. Single and multi-arm implementations were to be developed, along with evaluations of their performance.

This report will present in detail the implementation of the PAC force control functions into the testbed system. Characteristics of position-based force control

will be discussed, and some basic problems mostly overlooked in current literature will be identified and analyzed.

1.2 Brief Historical Review

The subject of robotic force control has been a long studied topic, with almost as many proposed methods as there are researchers in the field. Whitney[26] provides a historical review and classification of most force control methods developed to date. Hogan[8] proposes Impedance Control as an effective method of manipulation, and provides some comparisons to human manipulation. Maples and Becker[13] provided some experimental results of several different force control implementations. Hybrid Impedance has been discussed lately by Anderson[2]. The stability of robotic force control has been widely studied, and is discussed by An[1], Eppinger[5], and Lawrence[11].

Multi-arm manipulation has just recently been of interest, with several force control methods proposed by Wen[24, 25], Murphy[15], Kosuge[10], and Tao[20].

1.3 Report Organization

This report has been organized to provide a sufficient background on the theory of Position Accommodation, along with a description of the CIRSSE testbed, before discussing the implementation and performance of the PAC force control. With this goal in mind, the following chapters are outlined:

Chapter 2 discusses the theory of position-based force control and analyzes the force mechanisms involved. The PAC force control algorithms are presented and explained in detail.

Chapter 3 gives a description of the CIRSSE testbed, with sections on the Hardware and Software of the system. Emphasis will be on those systems directly related with the force control implementation.

Chapter 4 describes how the PAC force control algorithms were implemented into the testbed system. Details on the integration with trajectory generation will be discussed.

Chapter 5 presents the experiments performed with the PAC force control, and displays plots of the results.

Chapter 6 analyzes some of the characteristics of position-based force control, and discusses the performance of the PAC force control method. Problems faced when implementing the rotational compliance frames will also be discussed.

Appendix A contains a listing of the software used to implement the PAC force control algorithms.

CHAPTER 2

THEORY AND ANALYSIS

This chapter will discuss position-based force control in general, and will analyze some of the force mechanisms involved. The theory of Position Accommodation will be presented along with some of its physical concepts.

2.1 Position-Based Force Control

In position-based force control, the location of a manipulator's end-effector, or *tool-tip*, is used as the controlling variable in a force-feedback control system. Figure 2.1 shows the block diagram of a general position-based force control system.

Using a force sensor mounted on the end of the manipulator, the interaction forces between the manipulator and its environment, f_a , are feed back and summed with the desired force, f_d . The force error, f_e , is fed into a force-controller which passes a desired change in position, Δx_d , onto the position-controller of the manipulator. The position-controller will provide an actual change in position, Δx_a , which will in turn produce an interaction force through the force mechanisms present in the system.

2.1.1 Analysis of Force Mechanisms

In position-based force control systems, forces are generated by the compression of *spring-elements* present in the manipulation chain. Figure 2.2(a) depicts an ideal arm compressing a pure spring-element. The corresponding force developed, f , is simply found from the spring constant, k , and the displacement of the manipulator along the axis of the spring, Δx , as

$$f = \Delta x k \tag{2.1}$$

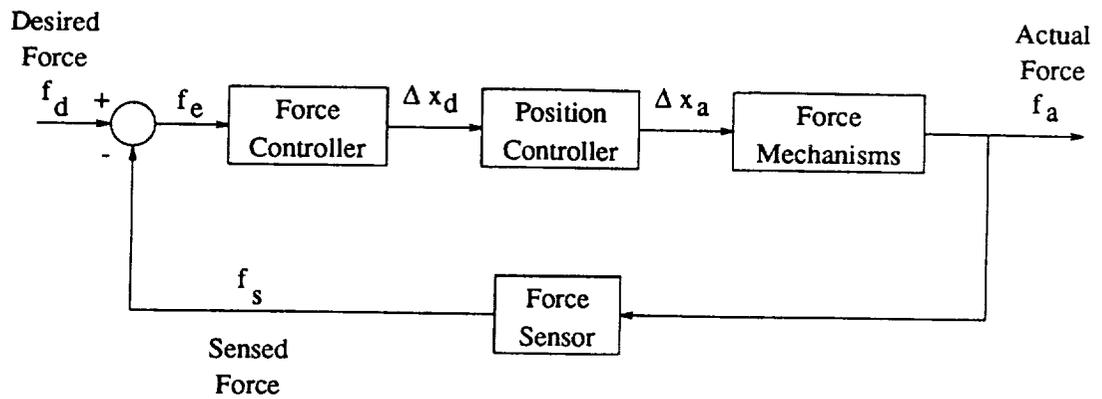


Figure 2.1: Position-Based Force Control

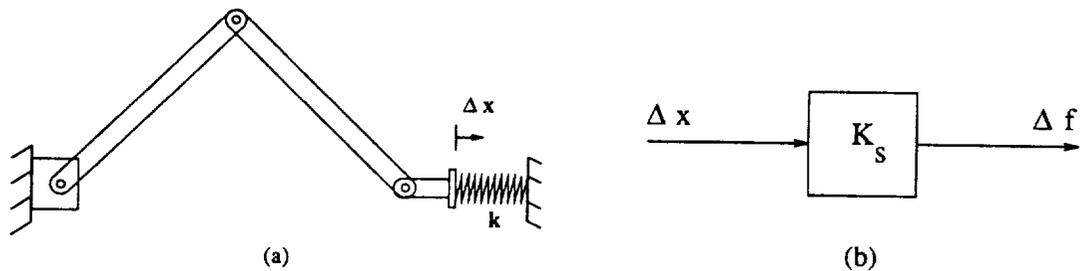


Figure 2.2: (a) Ideal Arm and Spring; (b) Force Function Block

Figure 2.2(b) depicts this relationship in block diagram form. This can be directly used as the **Force Mechanisms** block in Fig. 2.1. Thus, *the stiffness of the spring-elements present in the force control system directly effects the forward-gain of the force control loop*. The significance of this is that the stiffness of these spring-components is often unknown, and can dynamically change during manipulation. This can pose severe limitations on the maximum gains that can be used in the force control loop, thus limiting its performance[18]. Often, the force control gains are set relatively low to trade-off performance for stability.

There are two sources for the spring-components in a robotic manipulator:

1. *Physical-Stiffness* of manipulator linkages, force-sensors, workpieces, etc.

2. *Servo-Stiffness* of the manipulator's position-control system.

The Physical-Stiffness components are usually fairly large ($> 10^4$ [N/m]) as these components are designed to have minimal deflection during manipulation. The Servo-Stiffness depends on several factors including: (1) the gain of the position-controllers; (2) the type of position-controller; and (3) the configuration of the manipulator linkages.

Spring-components physically located in series can be lumped together into a single stiffness term by the *parallel* combination of their spring constants:

$$\begin{aligned}
 K_{Lumped} &= K_{PositionController} || K_{Linkages} || K_{ForceSensor} || K_{Environment} \\
 &= ((K_{PositionController})^{-1} + (K_{Linkages})^{-1} \\
 &\quad + (K_{ForceSensor})^{-1} + (K_{Environment})^{-1})^{-1}
 \end{aligned} \tag{2.2}$$

In static operation (i.e. no acceleration of the manipulator) the smallest spring component will dominate K_{Lumped} , and $K_s = K_{Lumped}$ in Fig. 2.2(b). The influence of these force mechanisms on the force control experiments will be illustrated in Chap. 5, followed with a more detail discussion of their effects in Chap. 6.

2.2 Position Accommodation

Position Accommodation (PAC) is a form of position-based force control where the manipulator's nominal position can be modified in such a way as to have the manipulator simulate a mechanical impedance to its environment. Thus, the "position" of the manipulator's *tool-tip* "accommodates" to forces exerted on it from the environment. This is also described as "compliant" manipulation. By using the manipulator's position-control system, the dynamics of the arm are effectively decoupled from the force control operation. This allows attention to be focused on the force control dynamics themselves.

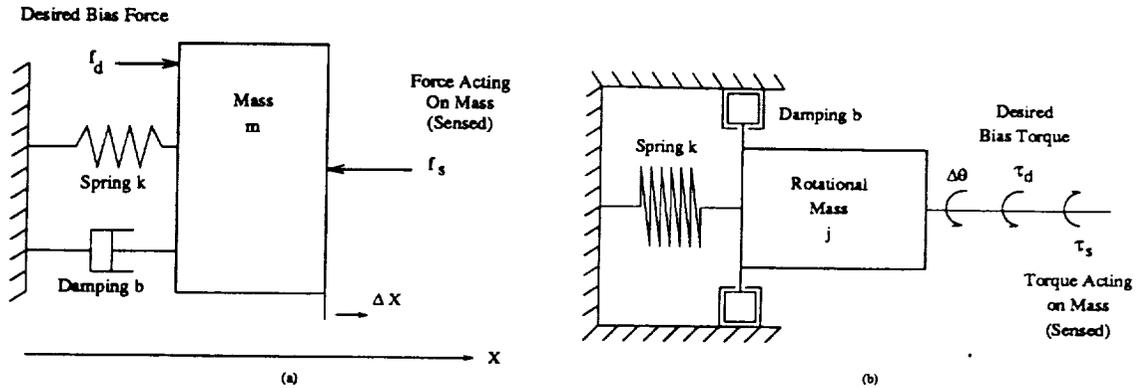


Figure 2.3: Mass-Spring-Damper Impedance Models: (a) Linear; (b) Rotational

With the assumption that we are only concerned with interaction forces at the grasp point, or *tool-tip* of the manipulator, forces and corresponding motions will be described in the *tool-frame* of the manipulator.

The impedance to be simulated by the manipulator is most commonly represented as a mechanical mass-spring-damper system. Figure 2.3(a) shows a linear model of such a system. The analogous rotational system is depicted in Fig. 2.3(b).

The differential equation of motion for the single-mass system shown in Fig. 2.3(a) is written as

$$m \Delta \ddot{x} + b \Delta \dot{x} + k \Delta x = \Sigma Forces = f_s + f_d \quad (2.3)$$

(Note: f_s as shown in Fig. 2.3(a) is negative). As an example, consider a manipulator under PAC force control, with one linear DOF along the x -axis. This manipulator would respond to forces felt along this axis just as the imaginary mass-spring-damper system in Fig. 2.3 would.

To facilitate compliant manipulation with an arm capable of six degrees-of-freedom, it is desired that the manipulator have the ability to comply to forces in all six *spatial* degrees-of-freedom (i.e. the manipulator will move in the direction of any applied force, and will rotate around the axis of any applied torque). This can

be referred to as “natural” motion for the manipulator because it is analogous to the motion exhibited by a passive mechanical system when acted upon by external forces (picture a bar of steel fixed at one end, with forces and torques applied to the other). To accomplish this, the forces seen in the tool-frame are broken down into their component parts ($f_x, f_y, f_z, \tau_x, \tau_y, \tau_z$), and applied to six versions of Eq. (2.3): one for each spatial degree of freedom. In matrix form, Eq. (2.3) can be re-written as

$$M \Delta \ddot{\mathbf{x}} + \beta \Delta \dot{\mathbf{x}} + K \Delta \mathbf{x} = \Sigma Forces = (\Phi \mathbf{f}_s + \mathbf{f}_d) \quad (2.4)$$

where:

M = The desired mass/inertia of the end effector

β = The viscous damping

K = The return spring force

Φ = The Phi-Matrix: transformation of forces/torques into the tool-frame

\mathbf{f}_s = The forces seen in force-sensor frame

\mathbf{f}_d = The desired, or bias, force in the tool-frame

$\Delta \mathbf{x}$ = The displacement vector, $\mathbf{x} - \mathbf{x}_{ref}$

The matrices M , β , and K are $[6 \times 6]$ diagonal matrices, where the $[i, i]$ element represents the mass, damping, or spring parameter, respectively, for the i th spatial axis. The Phi-Matrix, Φ , is necessary because the force-sensor is not physically located at the grasp point of the manipulator. Thus, forces sensed at the force-sensor must be transformed into the corresponding forces felt at the tool-tip. This will be discussed in Sec. 2.2.1. The position \mathbf{x}_{ref} represents the nominal location of the manipulator’s tool-tip with no force applied.

Equation (2.4) will produce six decoupled differential-equations for $\Delta \mathbf{x}$: three linear mbk systems and three rotational jbk systems. After integration, the six equations will produce a displacement vector in the *impedance space* of the manipulator

in the form of:

$$\Delta \mathbf{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \alpha \\ \Delta \phi \\ \Delta \theta \end{bmatrix} \quad (2.5)$$

where $\Delta \alpha$, $\Delta \phi$, and $\Delta \theta$ represents rotational displacements around the x , y , and z axis, respectively, of the tool-frame.

Since the forces and torques are sensed in the *cartesian space* of the manipulator's tool-frame, the displacement vector of Eq. (2.5) must be converted into a homogeneous transform[7] representing the cartesian displacement of the manipulator in its tool-frame. This transform will be referred to as T_{Δ} . For small-angle displacements, the order of rotation will make little difference on the response of the manipulator. When applied torques produce large rotations, the order of rotation is quite significant, and different orderings can produce drastically different results. This fact of implementation has received little attention in the literature. For the force control experiments performed in this project, the following sequence of rotations and translations were chosen for the conversion:

1. Yaw around the tool-frame x-axis by $\Delta \alpha$.
2. Pitch around the tool-frame y-axis by $\Delta \phi$.
3. Roll around the tool-frame z-axis by $\Delta \theta$.
4. Translate along tool-frame x-, y-, and z-axes by Δx , Δy , and Δz , respectively.

Combined, these form a homogeneous transform as

$$T_{\Delta} = T_{\Delta x \Delta y \Delta z} R_{Z, \Delta \theta} R_{Y, \Delta \phi} R_{X, \Delta \alpha} \quad (2.6)$$

This was chosen as the most straight-forward way to implement the Position Accommodation algorithm. The actual implementation will be discussed in Chap. 4. The limitations of this method of implementing compliant rotations will be discussed in Chap. 6.

2.2.1 The Phi-Matrix

For most implementations, the force/torque sensor is *not* located at the grasp point of the manipulator. Typically, a six degree-of-freedom F/T sensor is embodied in a strain-gauge bridge mounted between the flange of the robot and the gripper mechanism. Figure 2.4(a) depicts how the F/T sensor was mounted on the PUMA arms used in the experiments.

For compliant motion, it is desired that the manipulator act on forces/torques seen at the tool-tip frame, T_t . The F/T sensor will produce readings for forces/torques seen about its coordinate frame, T_s . Phi-Matrices[15] are used in general to translate forces and torques seen at one frame of a rigid-link to another. Here, the rigid-link is the combination of the gripper and the F/T sensor. Figure 2.4(b) shows the reference frames associated with the F/T sensor and the gripper. The homogeneous transformation from the T_t frame to the T_s frame is defined here as

$${}^tT_s = \begin{bmatrix} {}^tR_s & {}^t\mathbf{p}_{t,s} \\ 0 & 1 \end{bmatrix} \quad (2.7)$$

For a force sensed in the T_s frame, f_s , the force in the T_t frame, f_t , is found by simply rotating the force back to the T_t frame:

$$f_t = {}^tR_s f_s \quad (2.8)$$

The torque felt in the T_t frame, τ_t , will be the rotated torque sensed in the T_s frame, τ_s , summed with the lever-arm torque induced by the forces sensed in the T_s frame:

$$\tau_t = {}^tR_s \tau_s + {}^tR_s ({}^s\mathbf{p}_{t,s} \times f_s) \quad (2.9)$$

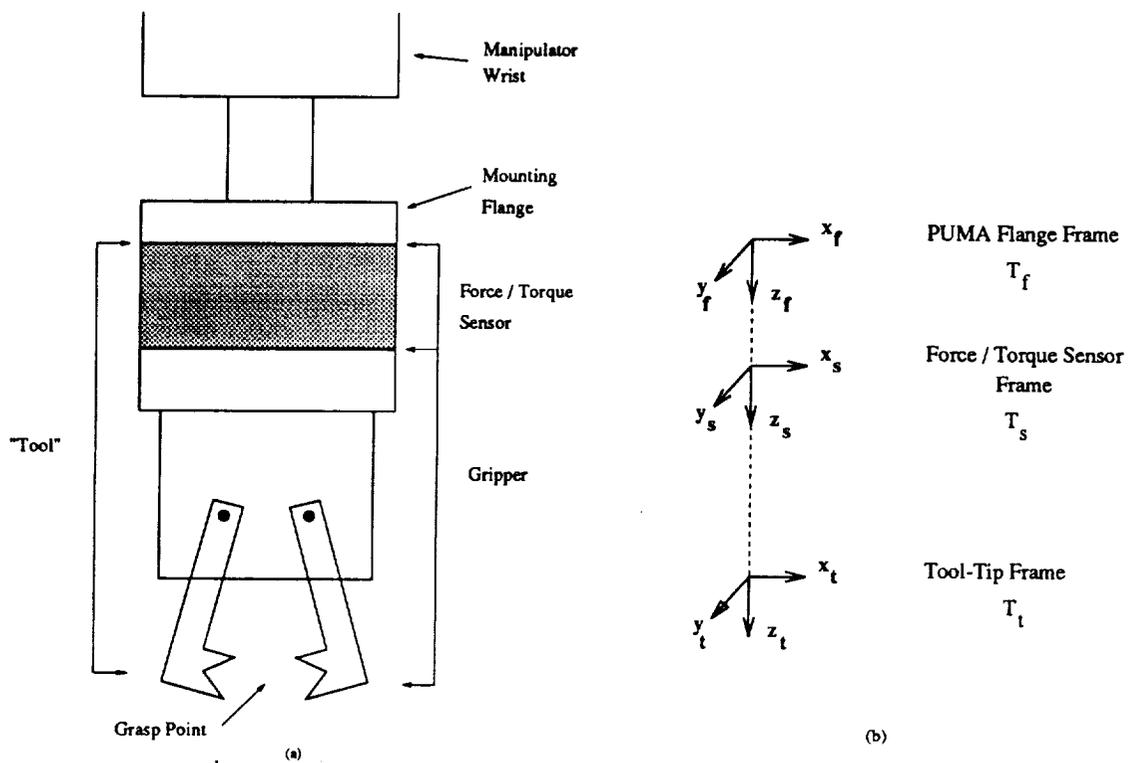


Figure 2.4: (a) Mounting of Force/Torque Sensor; (b) Kinematic Frames of Flange, Sensor, and Tool-Tip

The cross-product of ${}^s\mathbf{p}_{t,s}$ with f_s may be represented in matrix form as

$$({}^s\mathbf{p}_{t,s} \times f_s) = {}^s\tilde{P}_{t,s}f_s, \quad (2.10)$$

where

$${}^s\tilde{P}_{t,s} = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix} \quad (2.11)$$

The terms $[p_x \ p_y \ p_z]$ in Eq. (2.11) are the components of ${}^s\mathbf{p}_{t,s}$. The term ${}^s\tilde{P}_{t,s}$ may be derived from ${}^t\tilde{P}_{t,s}$ by

$${}^s\tilde{P}_{t,s} = {}^sR_t {}^t\tilde{P}_{t,s} {}^tR_s \quad (2.12)$$

where ${}^t\tilde{P}_{t,s}$ is found from tT_s as in Eqs. (2.10) and (2.11).

Combining Eqs. (2.9), (2.11) and (2.12) will yield the torque in the T_t frame as

$$\tau_t = {}^tR_s\tau_s + {}^tR_s {}^sR_t {}^t\tilde{P}_{t,s} {}^tR_s f_s \quad (2.13)$$

$$= {}^tR_s\tau_s + {}^t\tilde{P}_{t,s} {}^tR_s f_s \quad (2.14)$$

The Phi-Matrix operates on a stacked vector of forces and torques defined as (see Eq. (2.5))

$$\mathbf{f} \equiv \begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \quad (2.15)$$

To transform this six-vector of forces from one frame to another, the Phi-Matrix is formed from Eqs. (2.8) and (2.14) as

$$\Phi \equiv {}^t\phi^T(t,s) = \begin{bmatrix} {}^tR_s & {}^tP_{t,s}^{\sim} {}^tR_s \\ 0 & {}^tR_s \end{bmatrix}^T = \begin{bmatrix} {}^tR_s & 0 \\ {}^tP_{t,s}^{\sim} {}^tR_s & {}^tR_s \end{bmatrix} \quad (2.16)$$

Thus, to determine the forces/torques in the tool-tip frame (\mathbf{f}_t) given the forces/torques sensed in the F/T sensor frame (\mathbf{f}_s), the Phi-Matrix is used as

$$\mathbf{f}_t = {}^t\phi^T(t, s)\mathbf{f}_s = \Phi\mathbf{f}_s \quad (2.17)$$

2.3 Conclusion

This chapter has presented an overview of position-based force control, and has analyzed some of the force mechanisms involved. The theory of Position Accommodation has been presented along with some of the details involved with its implementation. The next chapter will describe the hardware and software systems of the CIRSSE testbed onto which the PAC force control was implemented.

CHAPTER 3

TESTBED DESCRIPTION

This chapter will describe the CIRSSE robotic testbed onto which the PAC force control algorithms were implemented. Descriptions of the robotic arms and computer-control systems will be given along with an overview of the software systems developed for the testbed.

The CIRSSE testbed was created to provide an experimental base for the development of cooperative robotic systems[4]. Of prime interest are assembly tasks where two robotic arms work together (much as our own left and right arms) to perform complex assembly tasks. Original motivation was taken from the strut and node assembly tasks outlined by NASA for the construction of a space-station.

3.1 HARDWARE

The CIRSSE testbed is an integration of several robotic systems including mechanical linkages, electrical motors and drives, computer-control, vision, force sensing, etc. This scope of this section will be limited to those systems involved in force control.

Figure 3.1 depicts the CIRSSE testbed. Two PUMA 6 DOF robotic arms are mounted on a two 3 DOF transporter platforms. Together they provide a total of 18 degrees-of-freedom. Table 3.1 gives the range of motion for each joint in the system. The coordinate frames and arm-configurations of the testbed are detailed in [23]. The overall system has been designed for a *joint-level* interface such that any combination of PUMA and platform joints may be enabled and used in an experiment.

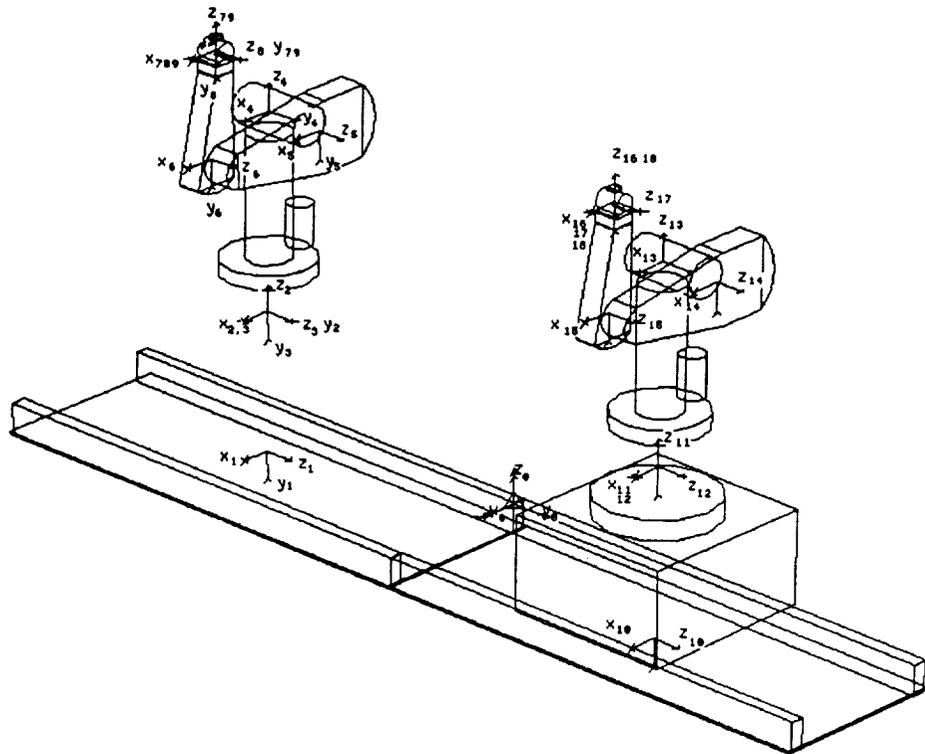


Figure 3.1: CIRSE Robotic Testbed

Table 3.1: Testbed Joint Parameters

Joint	Description	Range	Units
1	Left cart linear	-1524, 762	mm
2	Left cart rotate	-150, 150	degs
3	Left cart tilt	-45, 45	degs
4	Left PUMA shoulder	-250, 70	degs
5	Left PUMA upper-arm	-225, 45	degs
6	Left PUMA forearm	-45, 225	degs
7	Left PUMA wrist	-110, 170	degs
8	Left PUMA flange swivel	-100, 100	degs
9	Left PUMA flange rotate	-266, 266	degs
10	Left cart linear	-762, 1524	mm
11	Left cart rotate	-150, 150	degs
12	Left cart tilt	-45, 45	degs
13	Left PUMA shoulder	-250, 70	degs
14	Left PUMA upper-arm	-225, 45	degs
15	Left PUMA forearm	-45, 225	degs
16	Left PUMA wrist	-110, 170	degs
17	Left PUMA flange swivel	-100, 100	degs
18	Left PUMA flange rotate	-266, 266	degs

3.1.1 6 DOF PUMA Robots

The PUMA robots installed on the left and right sides of the testbed are Unimation models 560 and 600, respectively. These two models are functionally equivalent, and are mounted in identical fashion onto the transporter platforms.

The PUMAs are controlled by their original Unimation Controller boxes. The controllers have six Motorola-6502 based digital servo cards; one for each joint of the PUMA. These cards are mounted in a cage with a DEC¹ Q-Bus backplane. Each digital servo card commands a power-amp, which in turn drives a permanent-magnet DC joint motor. In addition, the digital servo cards interface with an encoder attached to the motor for joint position feedback. The power-amps are

¹Digital Equipment Corporation.

Table 3.2: Rated Capabilities of PUMA 560 Arm

Item	Specification	Units
Max Payload (including gripper)	22.3 (5.0)	N (lbs)
Static Force at Tool-tip	58 (13.0)	N (lbs)
Position Repeatability	± 0.1 (± 0.004)	mm (in)
Max Tool Acceleration	1	g
Max Tool Velocity	1.0 (3.3)	m/s (fps)

linear four-quadrant drives under current-loop control. Table 3.2 gives some of the rated capabilities of the PUMA 560 robot[22].

In controlling the joints of the PUMAs, the original VAL II control language used in the Unimation Controller is bypassed, and commands are sent directly to the digital servo cards. These cards can operate in two different modes:

Position Mode Position commands are sent to the digital servo cards every 28ms.

These are linearly interpolated down to 0.9ms, and summed with the position feedback from the encoders. The position error is passed through an analog PID controller to the power-amp's current-loop.

Torque Mode Torque commands are sent to the digital servo cards every 0.9ms.

These are scaled and sent directly out to the power-amp's current-loop. The encoder position can be read from the servo card in the same 0.9ms period.

The PAC force-experiments were run with position-controllers that utilized the *Torque Mode* of operation. These position-controllers are part of the Motion Control System described in Sec. 3.2.2. The dynamics of the PUMA 560 series arm are detailed in [16].

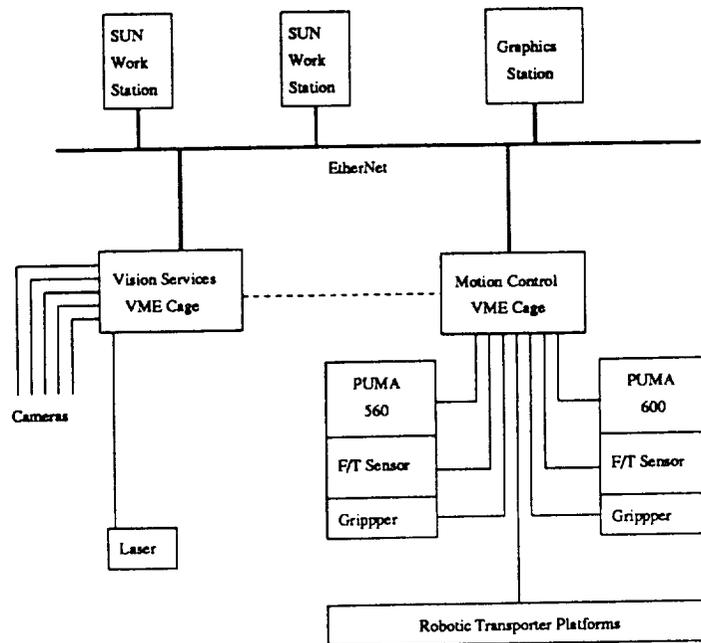


Figure 3.2: CIRSSE Testbed Computer System

3.1.2 3 DOF Transporter Platforms

The two transporter platforms of the testbed move along a linear rail mounted to the floor. The platforms and rail were manufactured by the K.N. Aronson company of Arcade, NY. The platforms were developed to extend the working range of the PUMA arms, and to give the testbed increased flexibility through joint-redundancy. The platforms provide linear, rotational, and tilt positioning of the PUMA arms. Details on the platforms and their control can be found in [3].

As with the PUMA arms, each joint of the platforms is driven by a DC motor connected to a power-amp. The power-amps have built-in current-loop control. Analog torque commands are sent directly to the power-amps from the testbed Motion Control System (see Sec. 3.2.2). Each joint motor has an attached encoder for joint position feedback.

Table 3.3: Modules used in VMEbus Cage

Pos.	Make and Model	Description
1	Motorola MVME-147SA-2	68030 Processor, 32MHz, 8 Meg Ram (Vx0)
2	Motorola MVME-224-1	Shared Memory Module
3	VMEbus to Q-Bus adapter	Left Unimate Interface
4	not used	
5	Motorola MVME-147SA-2	68030 Processor, 32MHz, 8 Meg Ram (Vx5)
6	Motorola MVME-135	68020 Processor, 16MHz, 1 Meg Ram (Vx1)
7	Motorola MVME-135	68020 Processor, 16MHz, 1 Meg Ram (Vx2)
8	Motorola MVME-135	68020 Processor, 16MHz, 1 Meg Ram (Vx3)
9	Motorola MVME-135	68020 Processor, 16MHz, 1 Meg Ram (Vx4)
10	Motorola MVME-340A	Parallel Interface/Timer Module
11	Whedco VME 3570-1	Dual Channel Encoder Interface
12	Whedco VME 3570-1	Dual Channel Encoder Interface
13	Whedco VME 3570-1	Dual Channel Encoder Interface
14	Motorola MVME-340A	Parallel Interface/Timer Module
15	Motorola DVME-628V	Digital to Analog Converters
16	not used	
17	VME Micro VMIVME-2532A	High Voltage Digital I/O
18	not used	
19	VMEbus to Q-Bus adapter	Right Unimate Interface
20	not used	
21	not used	

3.1.3 Computer-Control System

Figure 3.2 shows a picture of the distributed computer system developed for the CIRSSE testbed. At the heart of the system is a VMEbus[14] cage containing a variety of modules used in the control of the testbed systems. Table 3.3 gives a listing of the modules installed in the VMEbus cage at the time of this project. The distributed nature of the computer-control system provides the flexibility and performance needed to adequately control the various sub-systems of the testbed.

There are two VT320 terminals attached to the VMEbus cage: one connected directly to CPU module Vx0 and the other connected to a switch-box going to the other 5 CPUs (Vx1 - Vx5). These provide a user-interface to the control processes

Table 3.4: Force/Torque Sensor System

Make	Lord Industrial Automation		
Model	15/50		
Capacity	Force	15	lbs
	Torque	50	in-lbs
Resolution	F_x, F_y	0.174	oz
	F_z	0.576	oz
	T_x, T_y, T_z	0.391	in-oz
Frequency Response	Parallel Port	303	Hz

running on each CPU module. In addition, the VMEbus cage is connected to the CIRSSE computer network via an EtherNet gateway on CPU Vx0. This allows processes running on the VMEbus CPUs to interface with other processing platforms connected to the EtherNet, such as the SUN² workstations located in the testbed lab.

To interface directly with the 6502 digital servo cards in the Unimation Controller boxes, two VMEbus to Q-Bus adaptors were purchased. These facilitate the control of the left and right PUMA arms directly from the processors in the VMEbus cage[4].

3.1.4 Force Sensors

Two Force/Torque sensors provide force-feedback for the testbed system. These F/T sensors are mounted on the end of each PUMA arm, sandwiched between the mounting flange and the gripper (see Fig. 2.4(a)). They provide force and torque readings for all six spatial degrees-of-freedom. Table 3.4 lists some of the specifications of the F/T sensors[12].

Low-level control of the F/T sensor hardware is accomplished via serial ports

²SUN Microsystems, Mountain View, CA.

on two of the processors in the VMEbus cage. Force and torque data is accessed via a parallel interface between the F/T sensor hardware and a Parallel Interface/Timer (PIT) module in the VMEbus cage. This parallel data interface allows F/T data readings to be taken at 300Hz.

3.1.5 Pneumatic Grippers

On the end of each PUMA arm are mounted two pneumatic grippers. The grippers were custom designed specifically for the handling of model struts used in assembly experiments. The grippers use opposing air cylinders to control the opening and closing of the jaws. Linear potentiometers and strain-gauges provide feedback for position and/or force control of the gripper jaws. In addition, a cross-fire sensor in the jaws can be used to sense when a strut is between them.

Two gripper control units provide the interface hardware between the gripper mechanisms and CPU modules in the VMEbus cage[4]. Communication to the gripper control units is accomplished via serial ports on the processors. The actual servo control of the grippers is done through synchronous processes running on the VMEbus CPUs.

The mass and inertia parameters of the grippers have been characterized, and is detailed in [19].

3.2 SOFTWARE

The software developed for the CIRSSE testbed represents several man-years of design and development efforts. It was designed with the goal to have as flexible a system as possible without sacrificing performance.

The core of the system is the VxWorks³ multi-tasking operating system, which

³Wind River Systems, Alameda, CA.

runs on the Motorola processor modules in the VMEbus cage. This is a UNIX-like operating system designed for real-time micro-processor control systems. It has a several libraries of functions to support synchronous timing, inter-process communication, etc.

All the source code for the testbed is written in the C language[9]. Code development is done on SUN workstations running UNIX. Source code is cross-compiled with the GCC⁴ compiler to run on the Motorola processors located in the VMEbus cage.

To provide support for software tasks distributed across several processing platforms, the CIRSSE Testbed Operating System (CTOS) was developed as an extension to the VxWorks OS. To provide a clean, consistent interface to the testbed hardware, the Motion Control System (MCS) was designed and developed. These two software systems will be described briefly in the following sections.

3.2.1 CIRSSE Testbed Operating System (CTOS)

CTOS[6, 4] was designed to provide a real-time, homogeneous, distributed operating system in which processes used to control the testbed can communicate to the testbed hardware, and with other processes, regardless of which processing platform the process was running on. This allows developers to distribute their processes across the CPU modules in the VMEbus cage, or even to the SUN workstations on the CIRSSE EtherNet[17]. The obvious gain in processing power is somewhat offset by additional communication overhead.

The processor-independent nature of CTOS allows the distribution of processes on the VMEbus CPU modules to be rearranged without breaking the inter-process communication links. At boot-up time, a configuration file defines the distribution of the processes and support code.

⁴GNU project C Compiler, Free Software Foundation.

Highlights of the CTOS functionality include:

- A homogeneous message-passing system that provides a standard communication interface, regardless of the actual location of the source and destination processes. Though not “real-time”, the message-passing is sufficiently fast for most inter-process communication.
- A time-synchronization service that allows multiple processes across multiple CPU modules to be synchronized at different clock rates.
- A Shared Memory library for allowing several processes to access the same data at real-time speed, regardless of which CPU module they are on. This provides real-time system-wide access to joint-position data, F/T sensor data, etc.

3.2.2 Motion Control System (MCS)

The Motion Control System was designed to provide an architecture under which the various components of a robotic motion control system can be supervised[6]. Figure 3.3 depicts the architecture of the MCS system operating under CTOS.

Central to MCS is the State Manager. This function oversees the operation and inter-communication of the the processes involved in motion control. Before motion can begin, the individual components must “register” with the State Manager, where they will indicate which joints are to be used in an experiment. The State Manager ensures that the proper functions are in place before enabling the operation of a joint.

The Channel Drivers provide a clean interface to the motion hardware of the testbed. The function of the Channel Drivers is to pass torque commands from the motion Controllers out to the PUMA and Platform control hardware, and in return, read the joint position encoder values from the hardware and pass them back to the

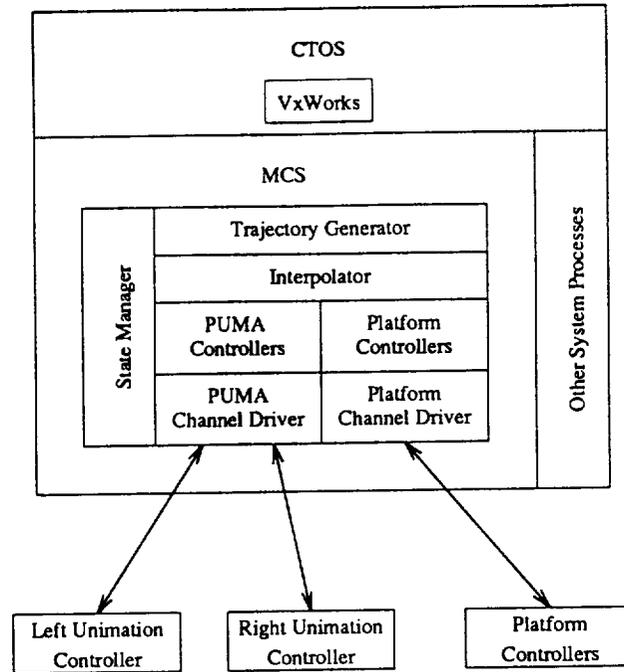


Figure 3.3: Architecture of MCS under CTOS

motion Controllers.

The PUMA and Platform controllers embody joint-level position control loops. Present controllers include PD, PID, and gravity-compensation. The PUMA controllers run at a whole-number multiple of the 0.9ms rate of the Unimation digital servo cards. The Unimation joint controllers are used in their Torque Mode of operation.

The Interpolators provide an asynchronous interface between the Trajectory Generator and the motion Controllers. Joint-vectors are sent to the Interpolators with an absolute time-stamp. The Interpolator uses this to correctly interpolate commands to the Controllers.

The Trajectory Generator provides an asynchronous interface between motion-planning and motion-execution. The TG takes as input files containing a list of joint-space “Knot-Points”[7]. It interpolates between these Knot-Points with blending functions[21] to control the transition time from one trajectory to the next. Details

of how the PAC force control was integrated with the TG will be given in the next chapter.

Data exchanges between the above motion control components takes place through the Shared Memory Module in the VMEbus cage. This enables the different components to be spread out over the CPU modules in the cage; providing a large advantage in processing power.

3.3 Conclusion

The CIRSSE robotic testbed was designed to be a high-performance, multi-purpose robotic system that will support a variety of research. Careful provisions have been made for the expansion and upgrade of various sub-systems over time. It is ideally suited for the robotic force control experiments in that it has an open architecture with consistent interfaces to the hardware of the testbed. The centralized control of the two robotic arms enables coordinated multi-arm experiments to be conducted. The distributed nature of the computer-control system, while adding some overhead, greatly increases the flexibility and processing power available for the execution of control algorithms.

The next chapter will review in detail the implementation of the PAC force control into the testbed.

CHAPTER 4

FORCE CONTROL IMPLEMENTATION

In the previous chapter, descriptions of the CIRSSSE testbed Hardware and Software systems were given. This chapter will focus on the implementation of the Position Accommodation (PAC) force control algorithms into these systems.

4.1 Trajectory Generation

Figure 4.1 depicts the joint-space Trajectory Generator (TG) discussed in Sec. 3.2.2. Joint-space Knot-Points are read asynchronously into the TG queue from a file. After interpolating and blending these points according to preset parameters of speed, acceleration-time, etc., joint-vector commands are sent at a periodic rate to the Interpolator. From there, they go on to the Controllers which servo the manipulator joints to the commanded positions. Table 4.1 lists the cycle-periods of the motion functions used for the force experiments.

The architecture of the Motion Control System was designed such that the TG could control from 1 to 18 joints, depending on what combination of PUMA and Platform joints are desired. As part of the MCS architecture, it was decided that any sensor-based path modification would take place at the TG level, as this would centralize all path-related functions.

4.1.1 Integration of Position Accommodation

In Chap. 2 it was shown that the PAC equations produce a homogeneous transform, T_{Δ} , that is to modify the nominal path of the manipulator. Since the TG operates in *joint-space*, forward-kinematics are required to convert the nominal joint-vector, θ , into a homogeneous transform suitable for multiplication with T_{Δ} . Correspondingly, inverse-kinematics are required to convert the modified transform

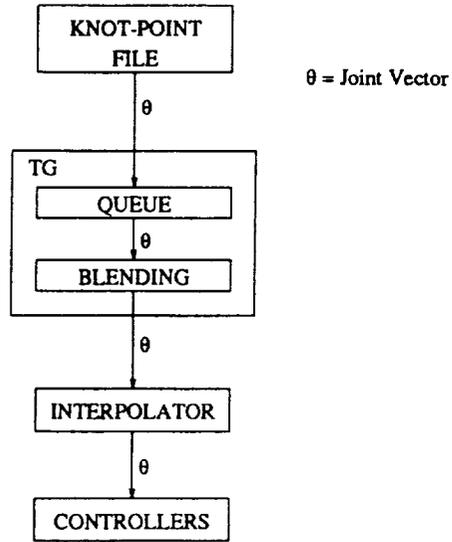


Figure 4.1: Trajectory Generator Data Flow

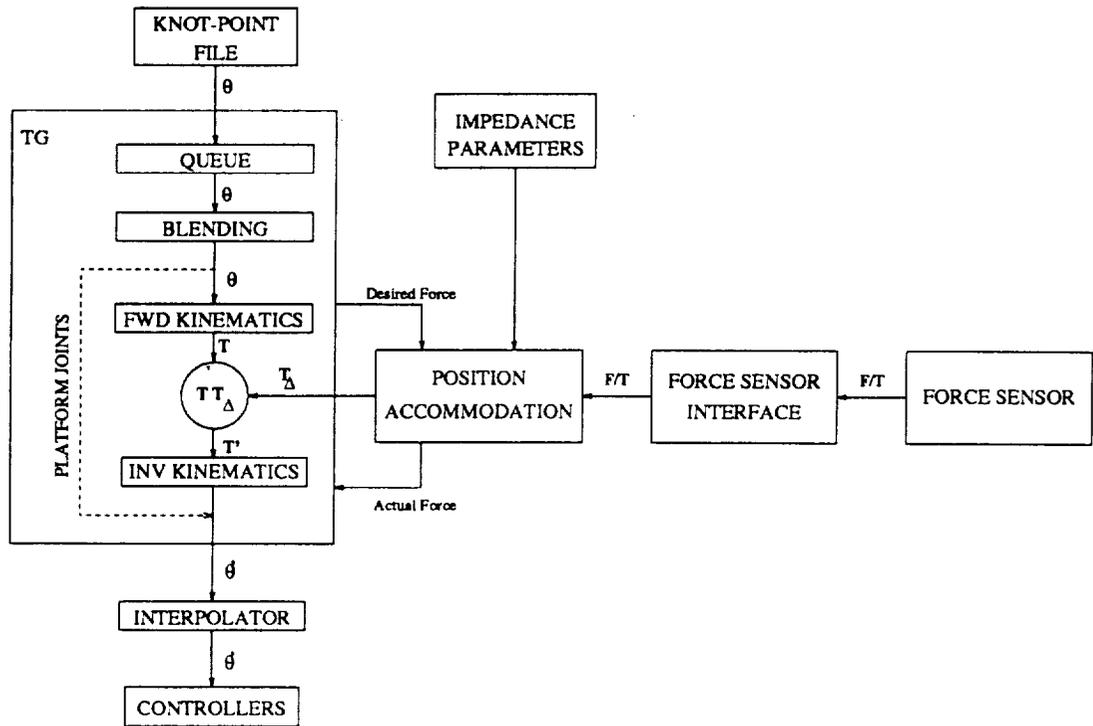


Figure 4.2: Trajectory Generator with Position Accommodation

Table 4.1: Cycle-Periods for Motion Control Tasks

Control Task	Cycle-Period	Units
Trajectory Generator	22.5	ms
PUMA Controllers	5.4	ms
Platform Controllers	5.4	ms

back into joint-space. Figure 4.2 depicts how the TG was altered to use the PAC path modification. Built into the PAC architecture is the ability to read in *Impedance Parameters* on-line from a data file. This greatly increases the flexibility of the PAC force control system as numerous experiments can be run sequentially without having to re-compile and re-boot the system.

To fully realize the cartesian path modifications of T_{Δ} , at least six joints of the PUMA–Platform combination must be activated. As force control experiments may be limited to use of a PUMA arm only, the 6 DOF forward- and inverse-kinematics of the PUMA were used to implement the path modification. Thus, it can be said that the PUMA arm performs the *force control* function, while the Platforms are used only for *motion*. This division was also influenced by the fact that forward- and inverse-kinematic routines were readily available for the PUMAs, while other routines involving the Platforms were still being developed.

Referring to Fig. 4.2, the blended joint-vector, θ , is passed into the forward-kinematics of the PUMA¹ arm to produce the nominal arm transform, T . Because the path modification is to take place in the *tool-frame* of the manipulator, this nominal transform is *post*-multiplied by T_{Δ} as:

$$T' = TT_{\Delta} \quad (4.1)$$

¹These kinematics account for the Platform location and give a transform with respect to the world coordinates of the testbed.

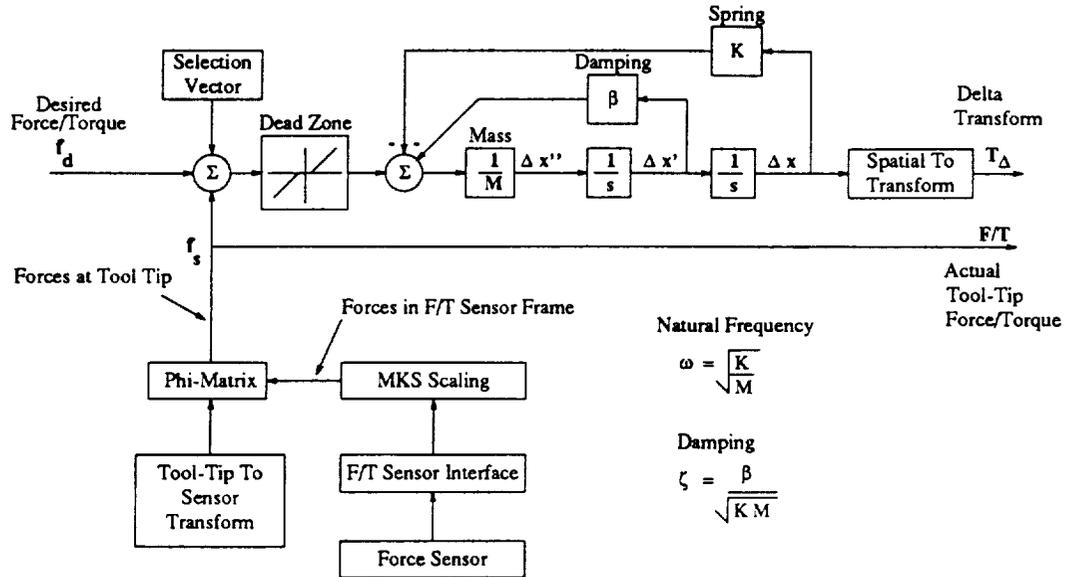


Figure 4.3: Position Accommodation: Continuous Model

The resulting transform, T' , represents the “Accommodated” position of the manipulator. This is passed through the inverse-kinematics of the PUMA to produce the modified PUMA joint-vector, θ' . The Platform and PUMA joint-vectors are then recombined and sent to the Interpolator.

The PAC function accepts as part of its parameter list a *desired-force* vector of the form shown in Eq. (2.15). This desired-force is used to implement *bias-forces* along the axes of compliance. Such bias-forces are used in some assembly tasks to “snap” components together. As an additional feedback, the *actual-force* seen at the tool-tip is passed back to the TG, where it can be used for data logging, force thresholding, etc.

4.2 Position Accommodation Function

Figure 4.3 shows the continuous model for the PAC function. The diagram follows the theory laid down in Sec. 2.2, with the following additional functionality:

- A *Selection-Vector* used to select which spatial degrees-of-freedom will be enabled for Position Accommodation.
- A *Dead-Zone* function for limiting the effects of noise, etc.

Both of these functions are directed by additional parameters located in the data file of Impedance Parameters.

The integrators depicted in Fig. 4.3 are implemented in the software using first-order rectangular integration. Figure 4.4 depicts the discrete implementation of the PAC function.

4.2.1 Tool to Sensor Transform

To properly transform the forces sensed at the F/T sensor to the tool-frame, a Phi-Matrix (see Sec. 2.2.1) derived from the tool-tip-to-F/T sensor transform, tT_s , is utilized. Included in the forward and inverse kinematic routines is a *tool-transform*, jT_t , which defines the gripper or tool mounted to the flange of the PUMA (refer to Fig. 2.4). Knowing this tool-transform, and the transform from the flange to the F/T sensor frame, jT_s , the tool-to-sensor transform is found as

$${}^tT_s = [{}^jT_t]^{-1} {}^jT_s \quad (4.2)$$

For proper operation, it is imperative that the tool-transform used by the TG correspond to the tool-to-sensor transform used by the PAC function. Without this, forces will be “sensed” at one point, and “accommodated” for at another, producing very unpredictable (and unnatural) motion. For this reason, the TG passes tT_s to the PAC function as part of its initialization. For single-arm experiments, the tool-transform was set to the center of the gripper jaws. Dual-arm tool-transforms will be discussed in Sec. 4.3.

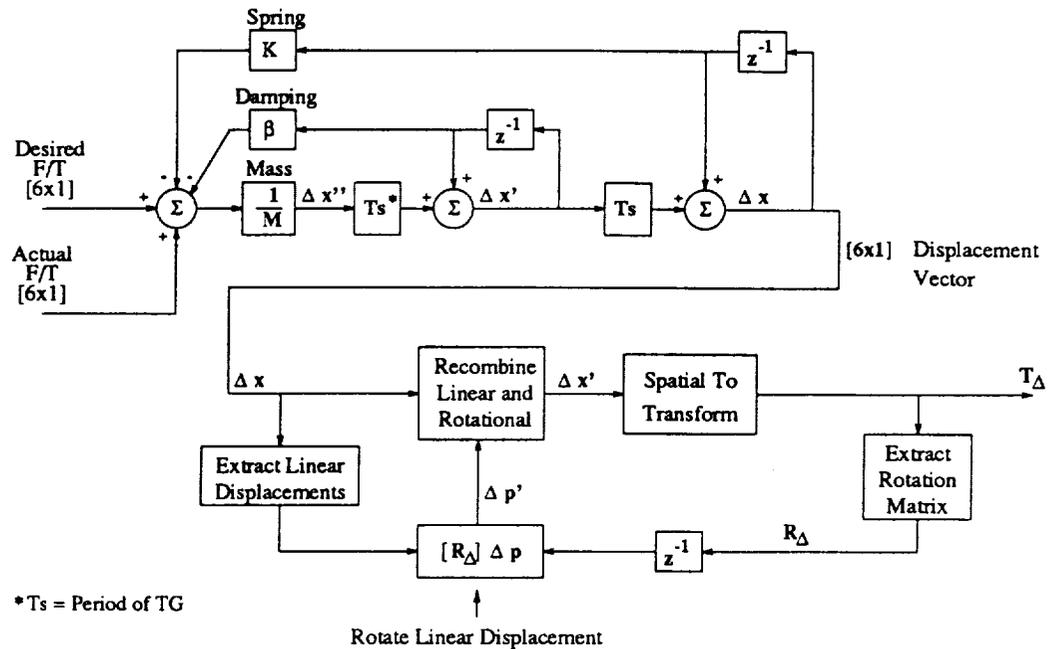


Figure 4.4: Position Accommodation: Discrete Model

4.2.2 Provision for F/T Sensor Rotations

When the manipulator is under PAC force control, rotations will move the axes of the F/T sensor away from their initial alignment with the nominal frame, T . When this conditions occurs, linear forces will be sensed in the F/T sensor-, or gripper-frame, but the PAC equations (see Eqs. (2.4) and (2.6)) will produce displacements in the nominal frame, thus leading to some very “unnatural” motion. To compensate for this, the linear displacements produced by the impedance equations were extracted from the displacement vector, $\Delta \mathbf{x}$, and rotated back into the original frame by using the rotational portion of the T_{Δ} transform computed in the last interaction. The assumption here is that the manipulator attained this rotation before the current F/T sensor readings were taken. Thus, the new displacement vector is found as

$$\Delta \mathbf{x}' = \begin{bmatrix} \Delta \mathbf{p}' \\ \Delta \mathbf{r} \end{bmatrix} \quad (4.3)$$

where

$$\Delta \mathbf{p}' = [R_{\Delta \text{previous}}] \Delta \mathbf{p} \quad (4.4)$$

Figure 4.4 depicts a discrete model of the PAC function with this linear-displacement rotation shown. The performance and limitation of this method of dealing with the rotations will be discussed in Chap. 6.

The source code for the PAC Function can be found in Appendix A, along with some support functions for handling the data files of Impedance Parameters.

4.2.3 Biasing of the F/T Sensor

Before the manipulator is placed into the PAC force control mode, the pre-existing forces registered by the F/T sensor must be biased out, as it is assumed that the impedance equations are initialized with zero applied-force. Two factors contribute to the non-contact forces seen by the F/T sensor:

1. The local *gravity field* acting upon the mass of the gripper and the F/T sensor.
2. *Internal* offsets in the strain-gauges and electronics of the F/T sensor.

The offsets internal to the F/T sensor are usually small (< 5%) and do not vary significantly over time or sensor orientation. The offsets due to the gravity field can be very large (depending on the mass of the gripper and payload), and will vary widely as the manipulator rotates with respect to the local gravity field. During an experiment, this gravity force-vector acts as an additional external force, and will cause the manipulator to “sag” downward. Currently, only a static biasing of the F/T sensor is performed prior to entering the PAC mode. Future versions of the F/T sensor interface function may include on-line gravity-compensation.

Since most of the force control experiments involve only small rotations through the local gravity field, the static biasing of the F/T sensor has been sufficient.

4.3 Dual Arm Implementation

When both arms are to manipulate a single object, kinematic errors will produce substantial linear and shear forces in the object as the arms move away from the initial starting position. This is especially true for the 2-arm manipulation of stiff objects. If the arms are under PID control, the joint torques would increase until the object and/or arms bent enough to account for the errors.

To avoid this problem, the PAC function is used with two 9-joint TGs: one for each of the left and right PUMA-Platform pairs. These TGs write joint-vectors to a common Interpolator, which in turn commands the Controllers for all 18 joints of the testbed. Figure 4.5 shows this dual-arm architecture.

When in the PAC force control mode, two separate tool-transforms will specify where the tool-tips of each manipulator will be located. This will directly effect the compliant motion during manipulation. Two schemes were found for locating the tool-tips of each manipulator:

1. The tool-tips of each arm were located at the *center of their grippers*. In this way, the arms behaved as they would under single-arm manipulation where the forces were sensed at the grasp point of the grippers, and consequential “compliance” was with respect to this grasp point.
2. The tool-tips of each arm were located at the *center of the strut*, i.e. the tool-tips *coincided*. The premise for this was that the arms would work together better when they were acting on the same point.

In the first scheme, motion files of Knot-Points for the two arms must take into account the shape of the strut. In the second case, the motion commands will refer to the same point, so one motion file for both arms could theoretically be used.

The performance of the dual-arm PAC force control will be discussed in Chaps. 5 and 6.

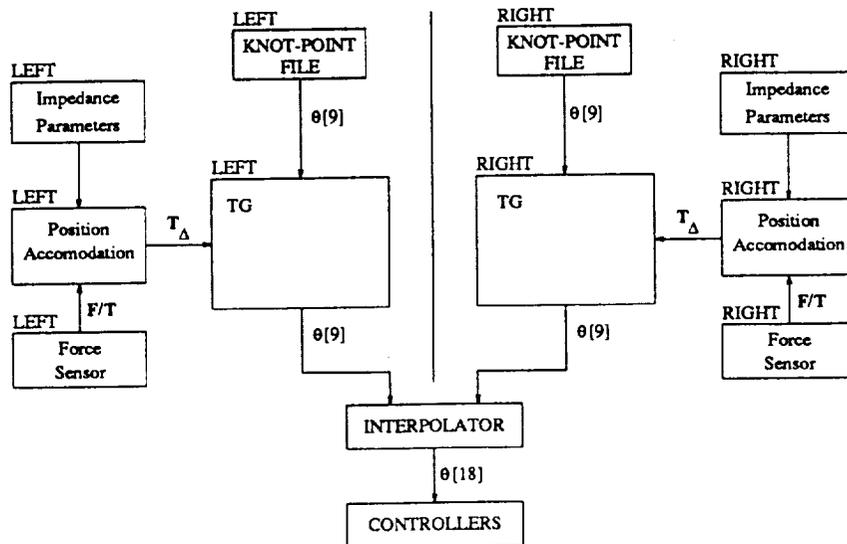


Figure 4.5: Dual Independent Trajectory Generators

4.4 Conclusion

This chapter has presented a detailed overview of how the Position Accommodation force control algorithms have been implemented into the CIRSSE testbed. Specific attention has been paid to the integration of the PAC function into the trajectory generation of the system. Continuous and discrete models of the PAC force control function have been given, along with a method for handling large rotations of the gripper under compliance. Dual-arm force control has been discussed, along with some options for its implementation.

The next chapter will present some of the experiments undertaken with the PAC force control method, along with various graphs of the results.

CHAPTER 5

EXPERIMENTS AND RESULTS

This chapter will present some of the experiments performed with the PAC force control, along with their results. The experiments fall into three basic categories:

1. Manipulator in free-air.
2. Manipulator contacting a fixed environment.
3. Two manipulators grasping a single object.

To simplify the analysis, most tests were performed along a single axis of compliance. In practice, all six degrees of compliance are typically used for assembly tasks, but the quantitative data from these tasks does not lend itself to a straightforward analysis (i.e. it is difficult to understand the interactions of all six degrees-of-freedom at once). Some tests with insertion tasks using the full 6 DOF compliance will be discussed from a *qualitative* point of view. In the section involving the manipulator contacting a fixed environment, a comparison will be made of the PAC force control performance with PD and PID position-controllers.

For all the tests, a 5.4ms position-controller was used with a 22.5ms trajectory generator.

5.1 Free-Air Tests

For this series of tests, the arm was placed in a nominal “safe” position as shown in Fig. 5.1, with no payload, excepting the gripper mass.

In the first test, the compliance was enabled for the linear z -axis only, with just damping and spring terms. A bias-force of 10N was commanded in the z -axis of the tool-frame (refer to Fig. 2.4). Figure 5.2 shows the linear-force and motion

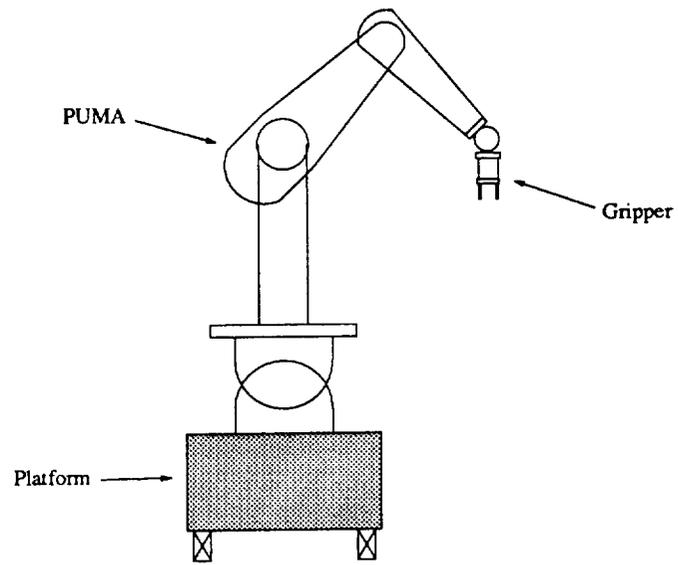


Figure 5.1: Manipulator in "Safe" Position

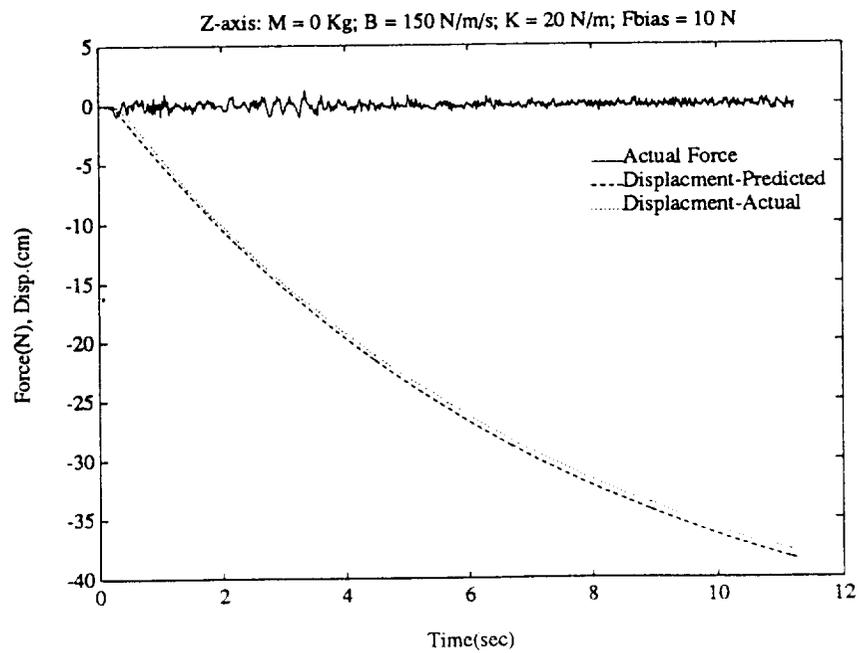


Figure 5.2: Linear Motion in Free-Air (Test1a)

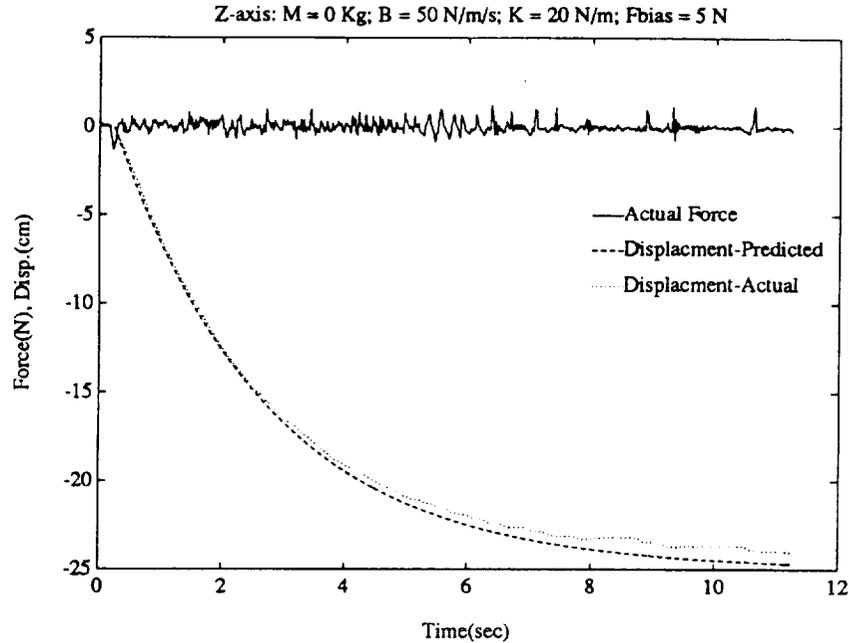


Figure 5.3: Linear Motion in Free-Air (Test1b)

of the manipulator. The motion of the arm has been measured in the *world-space* of the testbed.

As can be seen in Fig. 5.2, the arm moves down stretching the imaginary “spring” until the spring-force equals the bias-force. The damping directs the speed of this motion. Along with the actual motion is a curve depicting the predicted motion of such an impedance. This was simulated in MATLAB as a linear spring-damper system. From the graph it can be seen that the motion of the manipulator closely follows the predicted path. The actual-force plotted in Fig. 5.2 shows about $\pm 0.5 \text{ N}$ of noise. This is attributed to “jerking” in the arm motion causing accelerations of the gripper mass, thus producing an *inertial-force* on the gripper mass.

Figure 5.3 shows the same test repeated with slightly different damping and force terms. Notice that the position of the arm is settling out to approximately

-25cm, which corresponds with the expected final position:

$$\frac{5[\text{N}]\text{Force}}{20[\text{N/m}]\text{Spring}} = 0.25[\text{m}]\text{Displacement} \quad (5.1)$$

In Fig. 5.3 there can be seen some “jumping” of the position near the end of the plot. This has been attributed to friction in the PUMA joints, and will be discussed in Chap. 6.

5.2 Contact Tests

For these tests, the arm was again placed in a nominal “safe” position as shown in Fig. 5.1, with a cardboard box placed directly beneath the gripper. Cardboard has the characteristic of being fairly stiff, but will safely break away in case of a malfunction. As before, only the z -axis was enabled, and a bias-force was given such that the arm moved down to contact the box.

Figures 5.4 and 5.5 show two impact tests with a PD position-controller. Figures 5.6 and 5.7 show the same two tests with a PID position-controller. From the graphs in Figs. 5.6 and 5.7 it can be seen that the PID position-controller induces an oscillation in the force control when the manipulator comes in contact with the surface of the box. This effect will be discussed in Chap. 6.

5.2.1 Insertion Tests

The struts and nodes use in the CIRSSE testbed are rudimentary versions of the struts and nodes developed by NASA for constructing a space-station. The struts are designed to “snap” into spring-loaded clips at each node. Each node is designed to accept several struts spaced out by 120° . The strut and node assembly problem is detailed in [4].

During an insertion experiment, the manipulator grasps the strut in the middle and positions it over the target nodes located on a table-top. To insert the strut,

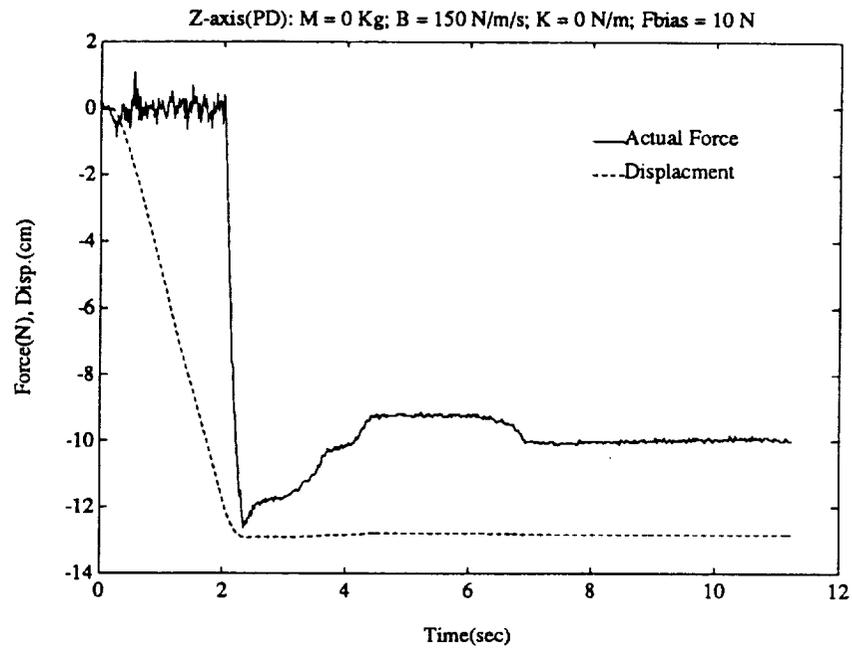


Figure 5.4: Impact with Box: PD Position-Control (Test2a)

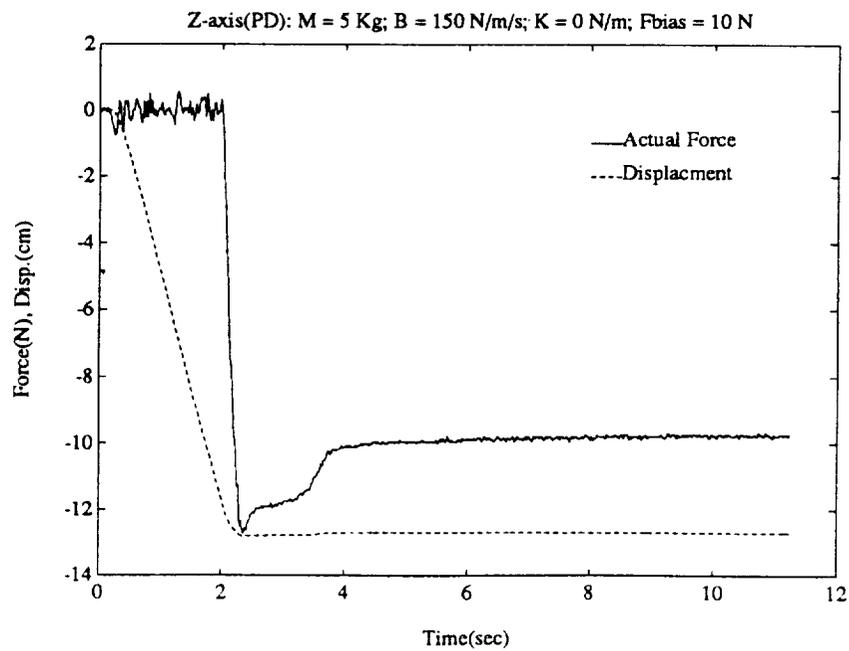


Figure 5.5: Impact with Box: PD Position-Control (Test2b)

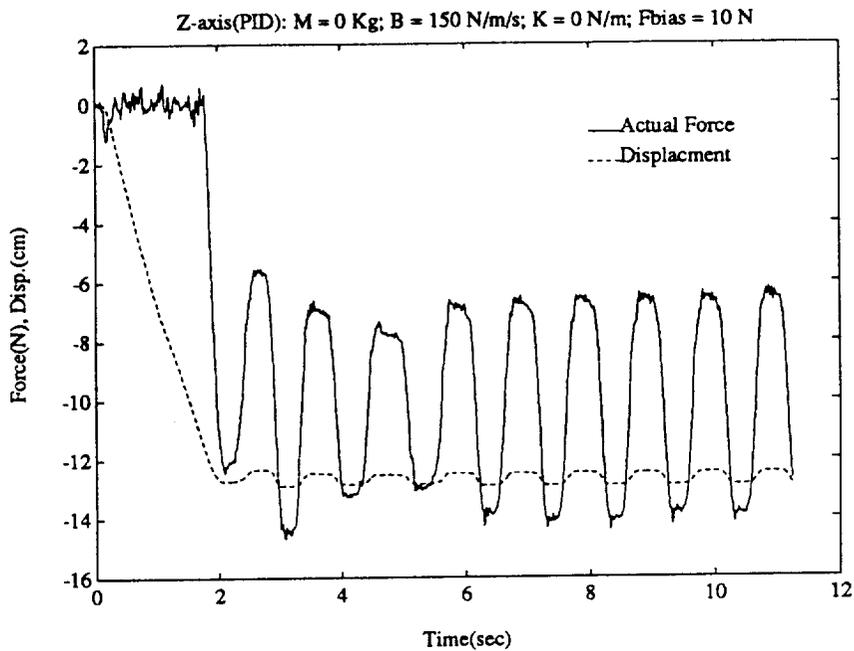


Figure 5.6: Impact with Box: PID Position-Control (Test2c)

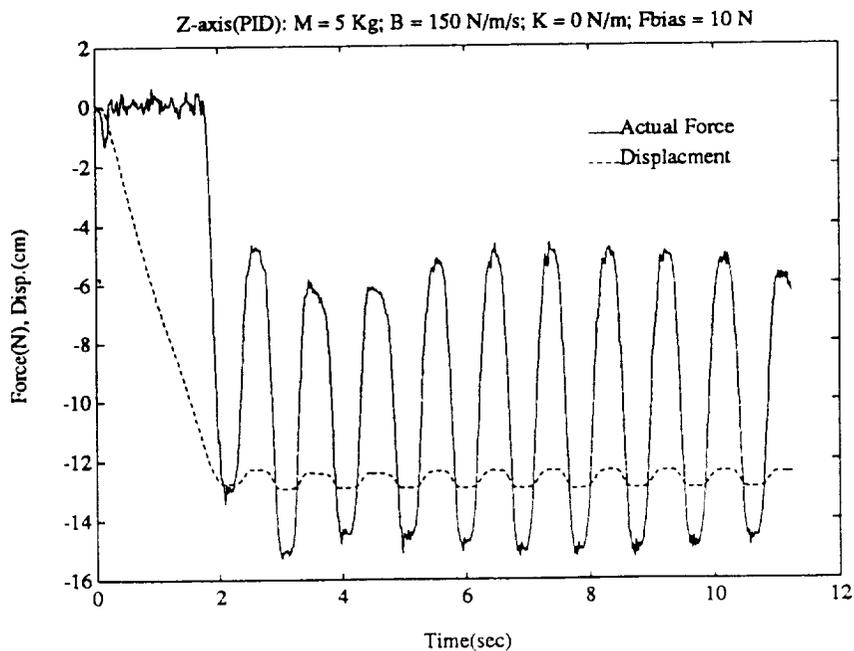


Figure 5.7: Impact with Box: PID Position-Control (Test2d)

Table 5.1: Insertion Impedance: PD Position-Controller

Parameter	x_{axis}	y_{axis}	z_{axis}	Units
Mass(linear)	90	90	100	kg
Mass(rotational)	0	0	0	kg*m
Damping(linear)	360	360	7500	N/m/s
Damping(rotational)	15	15	15	N*m/rad/s
Spring(linear)	180	180	0	N/m
Spring(rotational)	10	10	10	N*m/rad
Bias-Force(linear)	0	0	30	N
Bias-Torque(rotational)	0	0	0	N*m

Table 5.2: Insertion Impedance: PID Position-Controller

Parameter	x_{axis}	y_{axis}	z_{axis}	Units
Mass(linear)	180	180	100	kg
Mass(rotational)	0	0	0	kg*m
Damping(linear)	720	720	2500	N/m/s
Damping(rotational)	50	50	25	N*m/rad/s
Spring(linear)	360	360	0	N/m
Spring(rotational)	15	15	15	N*m/rad
Bias-Force(linear)	0	0	10	N
Bias-Torque(rotational)	0	0	0	N*m

the PAC force control is enabled with a bias-force to push the strut into the spring-clips. For these insertions all six degrees of compliance were used. Table 5.1 gives an example of the impedance parameters used in an insertion with a PD position-controller. Insertions performed with a PID position-controller exhibited noticeable oscillations, and the corresponding impedance parameters (Table 5.2) used were much more heavily damped.

To determine when an insertion was complete, the actual-force seen at the tool-tip was monitored, and the insertion was assumed finished when the actual-force settled to within a small percentage of the desired bias-force. Visual inspection

would indicate if the strut had completely missed the node spring-clips.

While the oscillations seen with a PID position-controller would seem at first glance to be an undesirable behavior, it was observed that these oscillations actually helped to “jiggle” a slightly misaligned strut into the spring-clips. In addition, because the PID controllers have better positioning accuracy than the PD controllers, strut alignment over the nodes was usually better with the PID controller.

5.3 Two-Arm Tests

For these series of tests the two PUMA arms were positioned to grasp a 0.7m-long steel strut by both ends. The arms were in their “safe” configuration (see Fig. 5.1) with the Platforms moved together to grasp the strut. Both arms were put under PAC force control with identical impedance parameters. Bias-forces, when applied, were set in opposing directions for the two arms such that a tension- or compression-force was applied to the strut. The tool-tips of each arm were located at the center of their gripper jaws.

Figure 5.8 shows the internal force, compression, and absolute position of the strut with no bias-force applied. For the most part, the internal force of the strut corresponds exactly with the compression of the strut, as expected. Large force “spikes” can be seen when the absolute position of the strut accelerates (change in slope), inducing an *inertial force* on the strut. The “drifting” exhibited by the absolute position of the strut was observed in most of the two-arm tests, and was attributed to the interaction between the PAC force-controllers of each arm.

Figures 5.9 and 5.10 show the same force and position parameters for a strut under compression and tension, respectively. For these tests, the compliance was restricted to only the x -axis of the tool-frame, along the length of the strut.

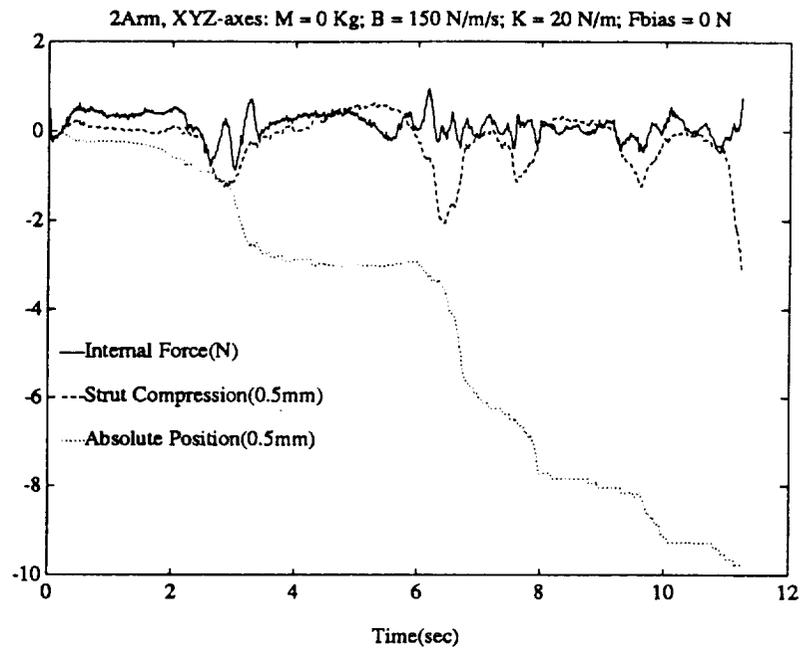


Figure 5.8: Two-Arm Test: No Bias-Force (Test3a)

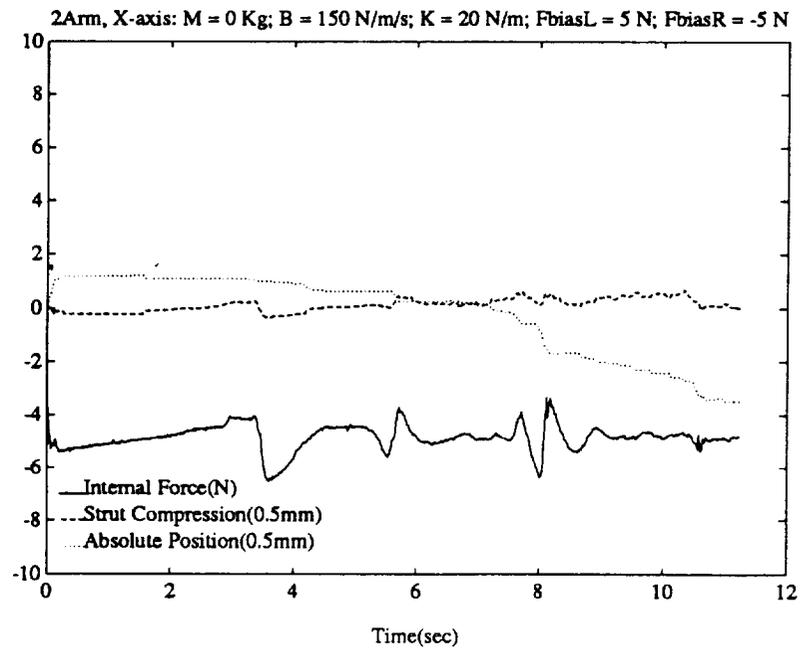


Figure 5.9: Two-Arm Test: Compression (Test3b)

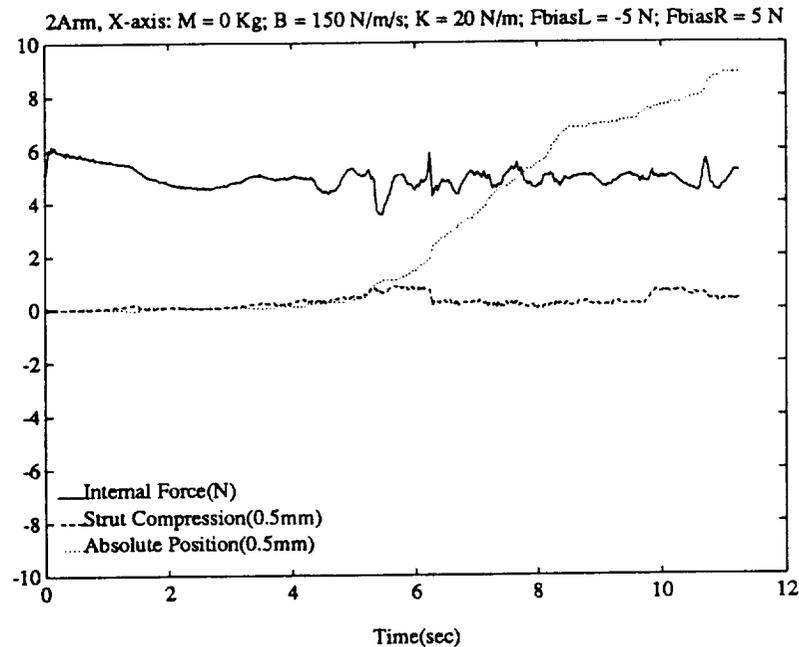


Figure 5.10: Two-Arm Test: Tension (Test3c)

5.4 Conclusion

The tests of the manipulator in free-air demonstrated that the PAC force control would accurately emulate a desired mechanical-impedance. This performance was degraded somewhat during slow motions, and this degradation has been attributed to the effects of joint friction/stiction on the position-controllers. It can be safely stated that *the performance of the PAC force control is directly dependent on the capabilities of the positioning system*. This will be true for all position-based force control systems.

When contacting a fixed environment, the PAC force control would correctly servo the arm to produce the desired bias-force. In comparing PD vs. PID position-controllers, it was found that the PID position-controller would induced large force and position oscillations when the arm came in contact with a stiff environment. For this reason, the PD position-controller is chosen for most tasks requiring PAC force control.

Numerous strut-insertion experiments have been performed with the PAC force control. For these insertions, all six degrees of compliance are enabled with a bias-force along the axis of insertion. While the PD position-controller produced a “smoother” insertion, the the oscillations present when using the PID position-controller would sometimes “jiggle” a slightly misaligned strut into the node spring-clips.

With the test involving two arms grasping a single strut, it was shown that the internal force of the strut can be controlled via the independent PAC force-controllers running on both arms. Interactions between the force-controllers would routinely produce a “drifting” motion of the strut. This “drifting” motion would induce an *inertial* force in the strut, which could be discerned in the force graphs.

The next chapter will discuss the characteristics of position-based force control, and review the performance of Position Accommodation force control as depicted in the experiments.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

This chapter will compare some of the theory and practice of robotic force control using Position Accommodation. The characteristics of position-based force control will be analyzed, and its comparative advantages and disadvantages will be listed. The performance of the PAC force control will be judged, along with some discussion of its limitations. Lastly, areas of further study will be identified.

6.1 Position-Based Force Control

Position-based force control was presented in Chap. 2. For most robotic systems, the feedback of joint positions is done via optical encoders. The resolution of these encoders determines the absolute *positioning resolution* of the manipulator. This positioning resolution will correspondingly determine a *force resolution* related to the force mechanisms discussed in Sec. 2.1.1. The significance of these force mechanisms will be discussed in this section, followed by a summary of the advantages and disadvantages of position-based force control.

Force Resolution Using some estimations, the linear force resolution, Δf_l , of the manipulator can be approximated from the linear position resolution of the manipulator, Δp_l , and the lumped linear stiffness of the manipulator's position-controller, force sensor, and gripper, k_l , as

$$\Delta f_l = \Delta p_l k_l \tag{6.1}$$

Table 6.1: Approximate Stiffness of Force Mechanisms: Z-axis

Component	Approx. Stiffness	Units
PD Position-Controller	10^1	N/m
PID Position-Controller	10^{10}	N/m
Force Sensor	10^4	N/m
Gripper	10^5	N/m

Table 6.2: Approximate Force Resolution: Z-axis

Parameter	PD Pos. Controller	PID Pos. Controller	Units
Lumped-Stiffness, k_l	99	9090	N/m
Force Resolution, Δf_l	0.01	0.9	N

Table 6.1 depicts the approximate linear stiffness the above components along their z -axis(see Fig. 2.4). These can be lumped together according to Eq. (2.2). Estimating the linear positioning resolution of the PUMA arm to be(see Table 3.2)

$$\Delta p_l \approx 0.1[\text{mm}] \quad (6.2)$$

the approximate linear force resolution for both PD and PID position-controllers can be found, and is shown in Table 6.2 along with the estimated lumped-stiffness. Note that the smallest stiffness dominates the lumped term, as would be expected.

As can be seen in Table 6.2, there is a significant difference in the lumped-stiffness and corresponding force resolution between the PD and PID position-controllers. It can be inferred that this would greatly affect the performance of the PAC force control and was demonstrated to do so in the experiments shown in Sec. 5.2. The experiments run with the PID controller showed sustained oscillations when contacting a fixed environment. These oscillations have been seen to persist even when heavily damped impedance parameters have been used, thus leading to

the conclusion that the oscillations may have been caused *not* by too much forward-gain, but rather represented a *limit-cycle* behavior due to the coarseness of the force resolution. This is one of the topics of further study discussed in a later section.

To avoid the oscillations seen in Sec. 5.2, a PD position-controller is used for most tasks requiring PAC force control. This brings out the quandary of position-based force control: for good positioning accuracy it is desired to have a very stiff manipulation mechanism(i.e. PID control), but this will correspondingly produce coarser force resolution. One option is to increase the positioning resolution, but because many robotic controllers use digital position encoders and digital control systems, there is a limit on how fine this position resolution can be. In practice, a PD position-controller is generally used, and stiff environments are avoided.

Advantages/Disadvantages Some advantage and disadvantages of position-based force control have been identified through this project and are listed below:

Advantages

- Easy to implement on existing manipulator control systems.
- Decouples the dynamics of the manipulator from the force control function.
- Integrates directly with trajectory generation.

Disadvantages

- The position-control system will have a lower bandwidth than the joint-torque controllers alone. Thus, a *direct-force*[13] control method which commands the joint torques directly will usually have the potential for higher performance.
- The discrete(digital) positioning characteristic of most position-control systems can lead to coarse force resolution. This can induce a limit-cycle behavior in the force control.

- The stiffness of the environment is an uncontrolled variable that directly affects the forward gain of the force control loop. Facing this unknown, force control performance is often traded for stability.

In spite of the above disadvantages, position-based force control appears to be a viable method of robotic force control, as long as provisions are made for the stiffness and force resolution of the manipulation system and its environment.

6.2 Performance of the PAC Force Control

Judging the performance of the PAC force control was hampered by the sheer complexity of its architecture: operating in all six degrees-of-freedom, it is difficult to quantitatively assess its performance. In a qualitative sense, the PAC force control performed quite well, especially when judged by its response to manually applied forces and torques. The response was very predictable and had a “natural” motion and feel to it. As the tests in Sec. 5.1 showed, the manipulator can accurately emulate a desired mechanical impedance, as long as the response necessary to emulate the impedance does not exceed the capabilities of the position-controller (i.e. only relatively damped impedances can be emulated).

The following sections will discuss some of the detailed performance characteristics of the PAC force control.

6.2.1 Force-Filtering

From the diagrams in Figs. 4.3 and 4.4, it can be seen that the PAC force control function acts as a *second-order filter* on the summed forces. This has the effect of filtering out any noise or large spikes present in the F/T sensor signal. If the PAC impedance is specified as only a damper and spring term, the system will act as a *first-order filter*.

When tests were tried with a spring-term alone, the system became wildly

unstable. This led to the conclusion that the filtering effect of the first and/or second-order impedances was necessary for stable operation. This limitation was not viewed to be a great handicap since spring-only impedances have little practical value.

6.2.2 Slow-Motions

The free-air experiment shown in Fig. 5.3 showed a slight “jumping” of the manipulator’s position when when the motion became relatively slow. This has been attributed to the position-controller’s response to stiction in the joints: the arm would slow to a stop, the friction would become relatively large, and the position error would build up until the arm “jumped” to its new position. This type of behavior has been seen for all types of manipulator motion where the velocity is relatively small.

The degradation of the position control during slow motions will produce a corresponding degradation in force control. The non-linear behavior of the joint-friction may also contribute to the oscillations observed when the PAC force control was implemented with a PID position-controller. Currently, the only option identified for improving the slow-motion performance is the use of faster position-controllers with higher proportional gains. This will require upgrading the processors and/or the architecture of the Motion Control System.

6.2.3 Implementation of Compliant Rotations

Perhaps one of the most avoided topics in robotic force control is the problem of implementing the compliant rotations of the full 6 DOF PAC force control. It is disturbing to see the extent to which the full 6 DOF theory of robotic force control has been developed in the literature with little or no recognition of the inherent limitations of implementing these rotations with actual manipulators and

force sensors.

As was outlined in Sec. 2.2, once a vector of spatial displacements($\Delta\mathbf{x}$) is produced, it must be converted into a homogeneous transform. For small rotations, any order of rotations will produce nearly identical responses, but once the rotations become significant several problems arise:

1. The F/T sensor is no longer aligned with the original “nominal” tool-frame. This was discussed in Secs. 2.2 and 4.2.2.
2. It is undetermined as to which frame the impedance parameters should follow; i.e. should the desired impedance be aligned with the original tool-frame, or should it follow with the gripper-frame?
3. When the impedance of a certain axis does not include a spring term, it is unclear what the motion of the manipulator should be when this axis displaces other axes with spring terms. In other words, it appears possible to specify a desired impedance which has *no physical equivalent*.

The implementation used in this project was outlined in Sec. 4.2.2, and has the following characteristics:

- Impedance parameters will follow with the *gripper-frame*, i.e. an impedance specified along the x -axis will remain with the x -axis of the gripper as it rotates. Bias-forces will also follow the gripper-frame.
- With the rotation sequence outline in Sec. 2.2, compliant rotations about any *one* of the gripper axes will work as expected. Compound rotations will only work properly when performed in the following sequence: (1) rotation about z ; (2) rotation about y ; (3) rotation about x .
- All rotations occur about the original “nominal” frame where the compliance

was initiated. Thus, large linear deviations from this origin will exhibit compliant rotations where the manipulator will “pivot” around the origin of the original frame.

- Rotation of the linear compliance terms, as illustrated in Fig. 4.4, will generally have linear motions occurring along the axis of the applied force (as would be desired).

Ideally, compliant rotations should be implemented in the order they are actually invoked on the manipulator. This could be accomplished if the “nominal” position of the manipulator was updated every control-iteration to the new position, and subsequent rotations were summed onto this position (i.e. every rotation would then be a relatively “small” rotation). Unfortunately, this method leads to problems in the manner in which the manipulator would “spring” back to its original position after having been linearly displaced and rotated.

Since most of the assembly tasks undertaken to date have not involved very large rotations, the manner in which the rotations are currently implemented has not seriously affected the force control performance. While a generalized method for complying “naturally” to any force/torque applied in all six degrees-of-freedom is not foreseen, specialized compliance schemes can be tailored to meet the requirements of almost any given task, as has been done with the implementation used for this project.

6.2.4 Limitations of PAC Force Control

The following identify some of the limitations encountered with PAC force control:

1. The PAC algorithms do not take into account the *inertial forces* connected with the masses of the F/T sensor, gripper, and payload. Fortunately, these have

- a “dampening” effect as the inertial force will oppose the current acceleration of the manipulator’s tool-tip.
2. Due to its dependency on the inverse-kinematics of the PUMA arm, the PAC algorithms will break down at the singularity points of the PUMA. Thus, motion that passes through a configuration change is not possible at this time. There are provisions in the kinematic code to work around this problem in the future.
 3. The effects of the local gravity field on the payload mass are not accounted for as the manipulator rotates this mass with respect to the gravity field. Some form of on-line gravity-compensation is planned as a future development for the PAC force control implementation.

6.2.5 Dual-Arm Manipulation

Section 5.3 depicted the results of several two-arm force control experiments. In general, the following was observed for two-arm PAC force control:

- The manipulators exhibit a “drifting” motion that appeared to be caused by the two PAC force-controllers “fighting” each other. This effect was diminished if only one or two axes of compliance was enabled, and heavily damped impedance parameters were used.
- Internal force control of compression and tension is possible, but is hampered by: (1) differentiating between *inertial* and *internal* force; (2) difficulties in force-servoing with stiff objects; and (3) determining whether the object is under tension or compression (this depends on the shape of the object and how it is grasped by the two manipulators).
- It is unclear how to handle large compliant rotations with dual-manipulators.

- It appeared that *coinciding* tool-tips worked slightly better than tool-tips centered at the grasp point of each manipulator. This is most certainly a qualitative judgment, and further research is needed to better understand the difference.

With the PAC force control running on each arm, a strut was successfully manipulated through several dual-arm paths. In addition, a dual-arm compliant insertion has also been accomplished. Overall, the PAC force control proved successful in enabling multi-arm manipulation to take place without passive compliance mechanisms or undue stress and strain on the manipulators and workpieces.

6.3 Future Work

If nothing else, this project has identified several areas of further study and development for Position Accommodation force control. The following outlines some of these areas:

- Further investigation into the oscillations observed when the manipulator contacts a stiff environment. Specifically, to determine the effect of *force resolution* on this behavior.
- Development of higher bandwidth position-controllers. This should directly improve the response and performance of the PAC force control.
- Further research into methods of applying compliant rotations, and of implementing the displacement vector, $\Delta\mathbf{x}$, into a path modification for the manipulator.
- Development of an on-line gravity-compensation scheme to cancel the effects of the local gravity field on the payload mass. This payload mass could be characterized at run-time by some simple rotations at the start of an assembly task.

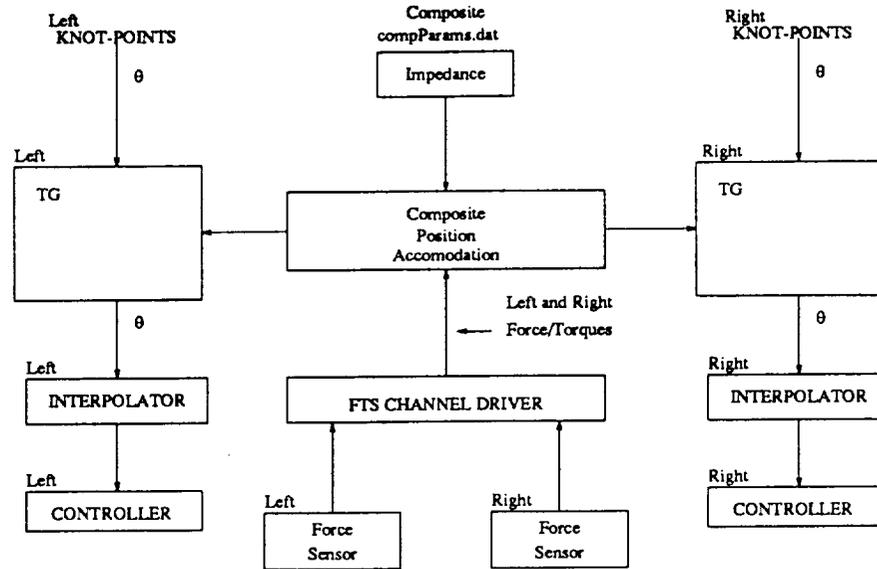


Figure 6.1: Dual Cooperative Trajectory Generators

- Development of a centralized dual-arm force control architecture. This would combine the forces from the left and right arms into a unified algorithm for controlling the internal as well as the net forces on an object. Figure 6.1 depicts such a system.

6.4 Conclusion

This project has thoroughly explored the implementation of Position Accommodation as a robotic force control method onto an 18 DOF robotic testbed. Several experiments have demonstrated the performance of this force control method with both single- and dual-arm implementations. Many practical considerations and limitations of this method of force control have presented that have not been previously covered in-depth in the literature. In addition, a review and discussion of position-based force control in general has been provided, along with some insights to the force mechanisms involved.

The PAC force control algorithms are presently installed as part of the software library of functions available for experiments on the CIRSSE testbed. These algorithms currently support several experiments with one- and two-arm assembly and manipulation tasks, and have significantly increased the scope of robotic tasks now possible with the testbed manipulators.

LITERATURE CITED

- [1] An, C.H., and Hollerbach, J.M., "Dynamic Stability Issues in Force Control of Manipulators", *Proc. 1987 IEEE Intl. Conf. on Robotics and Automation*, Raleigh, NC, pp. 890-896.
- [2] Anderson, R.J., and Spong, M.W., "Hybrid Impedance Control of Robotic Manipulators", *Proc. 1987 IEEE Intl. Conf. on Robotics and Automation*, Raleigh, NC, pp. 1073-1080.
- [3] Cosentino, J., "Development of a Control System for a Pair of Robotic Platforms", M.S. Thesis, ECSE Dept., Rensselaer Polytech. Inst., Troy, NY, August, 1990.
- [4] Desrochers, A.A., ed, *Intelligent Robotic Systems for Space Exploration*, Kluwer, Boston, 1992.
- [5] Eppinger, S.D., and Seering, W.P., "Understanding Bandwidth Limitations in Robot Force Control", *Proc. 1987 IEEE Intl. Conf. on Robotics and Automation*, Raleigh, NC, pp. 904-909.
- [6] Fieldhouse, K.R., et al, "Lecture Materials for the CTOS/MCS Introductory Course", Rensselaer Polytech. Inst., Troy, NY, CIRSSE Report #97, 1991.
- [7] Fu, K.S., Gonzalez, R.C., and Lee, C.S.G., *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, New York, 1987.
- [8] Hogan, N., "Impedance Control: An Approach to Manipulation, Parts I-III", *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, March 1985, pp. 1-24.
- [9] Kernighan, B., and Ritchie, D., *The C Programming Language*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1988.
- [10] Kosuge, K., et al, "Control of Single-Master Multi-Slave Manipulator System Using VIM", *Proc. 1990 IEEE Intl. Conf. on Robotics and Automation*, Cincinnati, OH, pp. 1172-1177.
- [11] Lawrence, D.A., "Impedance Control Stability Properties in Common Implementations", *Proc. 1988 IEEE Intl. Conf. on Robotics and Automation*, Philadelphia, PA, pp. 1185-1190.
- [12] Lord Corporation, *Installation and Operations Manual for F/T Series Force/Torque Sensing Systems*, Industrial Automation Division. Cary, NC, November 1987.

- [13] Maples, J.A., and Becker, J.J., "Experiments in Force Control of Robotic Manipulators", *Proc. 1986 IEEE Intl. Conf. on Robotics and Automation*, San Francisco, CA, pp. 695-702.
- [14] Motorola, Inc., *The VMEbus Specification*, October 1985.
- [15] Murphy, S.H., "Modeling and Simulation of Multiple Cooperating Manipulators on a Mobile Platform", Doctor of Philosophy Thesis, ECSE Dept., Rensselaer Polytech. Inst., Troy, NY, December, 1990.
- [16] Murphy, S., and Swift, D., "Dynamic Parameters and Inverse Dynamics for the PUMA 560", Rensselaer Polytech. Inst., Troy, NY, CIRSSE Tech. Memo #13, January, 1992.
- [17] Page, L., "Introduction to Using CTOS on Unix", Rensselaer Polytech. Inst., Troy, NY, CIRSSE Tech. Memo #16, April, 1992.
- [18] Roberts, R.K., Paul, R.P., and Hillberry, B.M., "The Effect of Wrist Force Sensor Stiffness on the Control of Robotic Manipulators", *Proc. 1985 IEEE Intl. Conf. on Robotics and Automation*, St. Louis, MO, pp. 269-274.
- [19] Swift, D., "Kinematic and Dynamic Parameters for the Testbed Grippers and Loads", Rensselaer Polytech. Inst., Troy, NY, CIRSSE Tech. Memo #14, January, 1992.
- [20] Tao, J.M., and Luh, J.Y.S., "Position and Force Controls for Two Coordinating Robots", *Proc. 1991 IEEE Intl. Conf. on Robotics and Automation*, Sacramento, CA, pp. 176-181.
- [21] Taylor, Russell H., "Planning and Execution of Straight Line Manipulator Trajectories", *IBM Journal of Research and Development*, Vol. 23, No. 4, July 1979.
- [22] Unimation, Inc., *Unimate PUMA Robot Manual*, Unimation Robotics, Danbury, CT, April 1980.
- [23] Watson, J., "Testbed Kinematic Frames and Routines", Rensselaer Polytech. Inst., Troy, NY, CIRSSE Tech. Memo #1, March, 1991.
- [24] Wen, J.T., and Murphy, S.H., "Stability Analysis of Position and Force Control for Robotic Arms", *IEEE Trans. on Automatic Control*, Vol. 36, No. 3, March 1991, pp. 365-371.
- [25] Wen, J.T., and Murphy, S.H., "Position/Force Control in Multiple-Manipulator Systems", *AIAA Space Programs and Technologies Conference*, March 1992, Huntsville, AL, AIAA 92-1676.

- [26] Whitney, D.E., "Historical Perspective and State of the Art in Robot Force Control," *Proc. 1985 IEEE Intl. Conf. on Robotics and Automation*, St. Louis, MO, pp. 262-268.


```

/*--- Include files needed for types, etc -----*/
#include "transLib.h"
#include "spatLib.h" /* for VECTOR6 types and functions */
#include "ftsLib.h"

/*--- FORCE_TORQUE_ENUM_TYPE -----
This typedef defines the order of reference of all 6-vectors used in
this package.
-----*/
typedef enum
{
    TRANS_X,      /* translation along x */
    TRANS_Y,      /* translation along y */
    TRANS_Z,      /* translation along z */
    ROTAT_X,      /* rotation around x */
    ROTAT_Y,      /* rotation around y */
    ROTAT_Z       /* rotation around z */
} FORCE_TORQUE_ENUM_TYPE;

/*---- IMPEDANCE_TYPE -----
The order of reference for each component in the structure follows the
FORCE_TORQUE_ENUM_TYPE listed above (i.e. the 4th element of the mass
component refers to the rotational mass (inertia) around the x axis).
-----*/
typedef struct _impedance_type
{
    VECTOR6
    mass,         /* Mass      [Kg]    and Inertia    [Kg*m]      */
    damping,      /* Damping-lin [N/m/s] and Damping-rot [N*m/rad/s] */
    spring;       /* Spring-lin  [N/m]   and Spring-rot  [N*m/rad]   */
} IMPEDANCE_TYPE;

/*--- Status Codes -----
These are the status codes for the position accomodation function.
*/

typedef enum
{
    PAC_FTS_OVERLOAD = 1, /* Impedance parameters are bad */
    PAC_FTS_ERROR = 2, /* Error using spatLib */
    PAC_BAD_IMPEDANCE_PARAMS = 3, /* Impedance parameters are bad */
    PAC_BAD_POINTERS = 4, /* improper pointer arguments */
    PAC_TRANS_ERROR = 5, /* Error using transLib */
    PAC_SPAT_ERROR = 6 /* Error using spatLib */
} PAC_STATUS_TYPE ;

/*-----

```

Function Prototypes

```

-----*/
/** function: pathModLibInit *****
|
| Description: General function for initializing the pathModLib.
|
| *****/
extern void
pathModLibInit();

/** function: posAcomReset *****
|
| Description: Resets the position accomodation integrators to zero.
|
| *****/
extern void
posAcomReset();

/** function: posAcomInit *****
|
| Description: This function sets the impedance of the position accomodation
| function (posAcomPathMod). The absolute value of all
| parameters is used; so negative values behave as would
| positive values.
|
| NOTE: This function also resets the state variables of the
| mass-spring-damper simulation back to zero.
|
| The select vector is a six-vector of ones and zeros used to
| select, or enable which axes of accommodation are to be
| activated. A value of (0) de-selects (disables) an axis;
| any value other than (0) (i.e. 1) selects (enables) an axis.
| The order of selection is defined in FORCE_TORQUE_ENUM_TYPE.
|
| The desired_F_T_dead_zone is a six-vector depicting the dead-
| zone to be implemented on the summed forces acting on the
| mass-damper-spring system.
|
| The tool2SensorTrans describes the transform from the desired
| compliant tool-tip to the force/torque sensor frame.
|
| *****/
extern PAC_STATUS_TYPE          /*ret: status          */
posAcomInit
( IMPEDANCE_TYPE *desired_impedance, /* in: impedance parameters */
  VECTOR6        *desired_select_vector, /* in: select vector        */
  VECTOR6        *desired_F_T_dead_zone, /* in: F & T dead zone      */
  TRANSFORM      *tool2SensorTrans, /* in: desired compliant tool tip */
  FTS_ID         desired_ftsId, /* in: Id of force sensor    */

```

```

float          trajGenPeriod          ); /* in: traj-gen period (secs) */

/** function: posAcomPathMod *****/
|
| DESCRIPTION: This function returns a delta-position transform given a
|               desired force vector. The actual (measured) arm force is
|               accessed using the chanFtsLib. The desired impedance is set
|               using the posAcomSetImpedance function.
|
|               The desired_force_torque vector is in accordance with the
|               FORCE_TORQUE_ENUM_TYPE order listed above.
|
| *****/
extern PAC_STATUS_TYPE          /*ret: status          */
posAcomPathMod
( VECTOR6   *desired_F_T, /* in: desired force[N]/torque[Nm]: tool-frame*/
  TRANSFORM *delta_trans, /*out: delta-transform          */
  VECTOR6   *actual_F_T ); /*out: actual force[N]/torque: tool-frame */

/** end of file: pathModLib.h *****/
#endif /* PATHMODLIB_H */

```

```

/* %% %G% */
/** File: pathModPrivate.h ****
|
|          NOTICE OF COPYRIGHT
|          Copyright (C) Rensselaer Polytechnic Institute.
|          1991 ALL RIGHTS RESERVED.
|
|
| Permission to use, distribute, and copy is granted ONLY for research
| purposes, provided that this notice is displayed and the author is
| acknowledged.
|
| This software was developed at the facilities of the Center for
| Intelligent Robotic Systems for Space Exploration, Troy, New York,
| thanks to generous project funding by NASA.
|
|-----
| Description: This file holds private variables, type, etc. for the sensor
|              based Path modification routines.
|
|-----
| --Rev---Date-----Author-----Description-----
| 0.1   10/21/91  MJ Ryan    Initial Release
| 0.2   10/25/91  MJ Ryan    Lowered force/torque sensor gains.
| 0.3   10/28/91  MJ Ryan    Fixed conversion gains, Added force/torque
|                               limits.
|
|-----
|-----/

#ifndef pathModPrivateh
#define pathModPrivateh

/*----- Constants -----*/

/*
** Constant for z-distance from end effector frame (at tool tip)
** to the force sensor frame:
*/

#define TOOL_TO_SENSOR_Z    (-0.150)          /* meters */

/* Constants for force and torque conversion */

#define F_CONV    (0.0556028 ) /* force: uf to Newtons */
#define T_CONV    (0.00141231) /* torque: uf*in to N*m */

/*----- Iinitial constants for impedance -----*/

```

```

/* Mass/Inertia */
#define M_X      (1.0) /* kg */
#define M_Y      (1.0)
#define M_Z      (1.0)
#define I_X      (1.0) /* kg*m */
#define I_Y      (1.0)
#define I_Z      (1.0)

/* damping */
#define D_X      (1.0) /* N/m/s */
#define D_Y      (1.0)
#define D_Z      (1.0)
#define D_R_X    (1.0) /* N*m/rad/s */
#define D_R_Y    (1.0)
#define D_R_Z    (1.0)

/* spring */
#define K_X      (1.0) /* N/m */
#define K_Y      (1.0)
#define K_Z      (1.0)
#define K_R_X    (1.0) /* N*m/rad */
#define K_R_Y    (1.0)
#define K_R_Z    (1.0)

/* selection definitions */
#define SELECT_1 (0)
#define SELECT_2 (0)
#define SELECT_3 (1)
#define SELECT_4 (0)
#define SELECT_5 (0)
#define SELECT_6 (0)

/* Default Time Period */
#define T_S      (0.040) /* defaults to 40ms */

/** end of file: pathModPrivate.h *****/
#endif

```

```

/* @(#)pathModLib.c 1.1 2/15/92 */
/**** File: pathModLib.c *****/
|
|          NOTICE OF COPYRIGHT
|          Copyright (C) Rensselaer Polytechnic Institute.
|          1992 ALL RIGHTS RESERVED.
|
|
| Permission to use, distribute, and copy is granted ONLY for research
| purposes, provided that this notice is displayed and the author is
| acknowledged.
|
| This software was developed at the facilities of the Center for
| Intelligent Robotic Systems for Space Exploration, Troy, New York,
| thanks to generous project funding by NASA.
|
|-----|
| Description: This file holds the function for the sensor based
|             Path modification routines.
|
|-----|
| --Rev--Date-----Author-----Description-----|
| 1.1  02/16/92 MJ Ryan      Initial Release      |
| 1.2  04/23/92 MJ Ryan      Fixed delay in posAcomPathMod, general cleanup |
|-----|
/*****/
/* TBD:
| - use ftsChanLib when ready
*/

#include "vxWorks.h"
#include "stdioLib.h"
#include "math.h"
#include "logLib.h"
#include "cirsse.h"          /* constants, etc */

#include "ftsLib.h"          /* force torque sensors */
#include "pathModLib.h"
#include "pathModPrivate.h"

/***** Constants *****/
const static VECTOR6
/* This is used to initialize state variables */
Zero_6 = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

const static TRANSFORM
identityTrans = { FASTX,
                 {1.0, 0.0, 0.0},
                 {0.0, 1.0, 0.0},
                 {0.0, 0.0, 1.0},
                 {0.0, 0.0, 0.0},

```

```

        {0.0, 0.0, 0.0},
        1.0           };

/*----- Static variables -----*/

/*
** This transform describes the transformation from the end effector
** frame (at the tool tip) to the force sensor frame. It is used with the
** routine spatPhiTransMult() in order to determine the forces felt at
** the tool tip.
*/

static TRANSFORM toolTipToSensorFrame =
{  FASTX,
   {1.0, 0.0, 0.0},
   {0.0, 1.0, 0.0},
   {0.0, 0.0, 1.0},
   {0.0, 0.0, TOOL_TO_SENSOR_Z},
   {0.0, 0.0, 0.0},
   1.0           };

/*-----
This transform will hold the rotation part of the previous deltaTrans
matrix. It is used to rotate the linear motions into the rotated frame.
-----*/
static TRANSFORM oldRotationTrans =
{  FASTX,
   {1.0, 0.0, 0.0},
   {0.0, 1.0, 0.0},
   {0.0, 0.0, 1.0},
   {0.0, 0.0, 0.0},
   {0.0, 0.0, 0.0},
   1.0           };

static IMPEDANCE_TYPE
/* the impedance is init to some stable mass-spring-damper */
impedance = { M_X, M_Y, M_Z, I_X, I_Y, I_Z,
             D_X, D_Y, D_Z, D_R_X, D_R_Y, D_R_Z,
             K_X, K_Y, K_Z, K_R_X, K_R_Y, K_R_Z };

static VECTOR6
/* these are the state variables for the delta position/rotation system */
delta_rate[2] = { {0.0,0.0,0.0,0.0,0.0,0.0},{0.0,0.0,0.0,0.0,0.0,0.0} },
delta_pos[2]  = { {0.0,0.0,0.0,0.0,0.0,0.0},{0.0,0.0,0.0,0.0,0.0,0.0} },

/* the select vector selects which axis of compliance are active */

```

```

select_vector = { SELECT_1, SELECT_2, SELECT_3,
                  SELECT_4, SELECT_5, SELECT_6 },

F_T_dead_zone = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0}, /* dead-zone params */

w_n           = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }, /* natural frequency */
zeta          = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }; /* damping ratio */

static float
  Ts          = T_S ; /* defaults to 40 ms */

static int
  system_order[6] = { 2, 2, 2, 2, 2, 2 } ; /* holds order of systems */

static FTS_ID
  ftsId       = FTS01422; /* initialized to right arm fts */

/** function: pathModLibInit *****
|
| Description: General function for initializing the pathModLib.
|
|---rev---date-----author-----description-----
| 0.2   10/26/91 MJ ryan      initial release
|*****
void          /* ret: void          */
pathModLibInit( void )
{

  /* TBD */
}
/** end of function: pathModLibInit ***/

/** function: posAcomReset *****
|
| Description: This function resets the position accomodation integrators to
|              zero.
|
|---rev---date-----author-----description-----
| 0.1   10/26/91 MJ ryan      initial release
| 0.2   11/08/91 MJ ryan      added reset of oldRotationTrans
|*****
void          /* ret: void          */
posAcomReset( void )
{
  int i;

  /* initialize state vectors to zero */
  for (i=0; i<2; i++)

```

```

{
    delta_rate[i] = Zero_6;
    delta_pos[i] = Zero_6;
}

/* initialize oldRotationTrans */
oldRotationTrans = identityTrans;

}

/** end of function: posAcomReset **/

/** function: posAcomInit *****/
|
| Description: This function initializes the parameters for the posAcomPathMod
| function. this includes:
|     the impedance parameters
|     the selection vector
|     the F_T_dead_zone
|     the tool-to-sensor-trans
|     the desired ft sensor ID
|     the trajectory generator period
|
|     NOTE: this function also resets the state variables of the
|     mass-spring-damper simulation back to zero.
|
|-----rev-----date-----author-----description-----
| 0.1  10/07/91  MJ Ryan    initial release
| 0.2  10/21/91  MJ Ryan    added state vector initialization
| 0.3  10/26/91  MJ Ryan    Added calculation of natural frequency and
|                               Damping
| 0.4  10/28/91  MJ Ryan    Added status return
| 0.5  10/29/91  MJ Ryan    Added tool2SensorTrans & tool2CompliantTrans
| 0.6  10/30/91  MJ Ryan    Formated printf output.
| 0.7  11/01/91  MJ Ryan    Added fts Id number to parameters
| 0.8  11/05/91  MJ Ryan    Added error checking for null pointers
| 0.9  11/11/91  MJ Ryan    Removed tool2CompliantTrans
|                               Added desired_F_T_dead_zone
|
| *****/
PAC_STATUS_TYPE          /* ret: status          */
posAcomInit
( IMPEDANCE_TYPE *desired_impedance, /* in: impedance parameters */
  VECTOR6        *desired_select_vector, /* in: selection vector      */
  VECTOR6        *desired_F_T_dead_zone, /* in: F & T dead zone      */
  TRANSFORM      *tool2SensorTrans, /* in: desired compliant tool tip */
  FTS_ID         desired_ftsId, /* in: Id of force sensor     */
  float          trajGenPeriod /* in: traj-gen period (secs) */
)
{
    int i;

    char

```

```

buffer1[80],
buffer2[80],
buffer3[80];

PAC_STATUS_TYPE
status = OK; /* Initialize to OK = 0 */

/* initialize local globals */
w_n = Zero_6;
zeta = Zero_6;

if (desired_impedance != NULL)
for (i=0; i<6; i++)
{
impedance.mass.v[i]      = fabs( desired_impedance->mass.v[i] );
impedance.damping.v[i]  = fabs( desired_impedance->damping.v[i] );
impedance.spring.v[i]   = fabs( desired_impedance->spring.v[i] );

if ( impedance.mass.v[i] != 0.0 )
{
system_order[i] = 2; /* mass-spring-damper */

/* find the natural frequency in Hz*/
w_n.v[i] =
( sqrt( impedance.spring.v[i] / impedance.mass.v[i] ) ) / PI2;
if ( impedance.spring.v[i] != 0.0 )
zeta.v[i] = impedance.damping.v[i]
/ ( 2.0 * sqrt( impedance.spring.v[i] * impedance.mass.v[i] ) );
}
else if ( impedance.damping.v[i] != 0.0 )
system_order[i] = 1; /* damper-spring */

else if ( impedance.spring.v[i] != 0.0 )
{
system_order[i] = 0; /* spring only */
logMsg("\007 \007 \007 WARNING!!!: SPRING ONLY IS UNSTABLE!!!");
}

else
{
logMsg("error: mass=spring=damper=0 ");
impedance.spring.v[i] = 10.0;
status = PAC_BAD_IMPEDANCE_PARAMS;
}

}/* end of for loop */
else
logMsg
("\007\007\007 WARNING!!!: posAcomInit: using default impedance!!!");

```

```

if (desired_select_vector != NULL)
  for (i=0; i<6; i++)
    if (desired_select_vector->v[i] == 0.0)
      select_vector.v[i] = 0.0;
    else
      select_vector.v[i] = 1.0;
else
  logMsg
    ("\007\007\007 WARNING!!!: posAcomInit: using default select vector!!!");

/* initialize state vectors to zero */
for (i=0; i<2; i++)
{
  delta_rate[i] = Zero_6;
  delta_pos[i] = Zero_6;
}

if (desired_F_T_dead_zone != NULL)
  F_T_dead_zone = (*desired_F_T_dead_zone);
else
  logMsg
    ("\007\007\007 WARNING!!!: posAcomInit: using default F_T_dead_zone!!!");

/* copy tool-tip to sensor frame transform */
if (tool2SensorTrans != NULL)
  toolTipToSensorFrame = (*tool2SensorTrans);
else
  logMsg
    ("\007\007\007 WARNING!!!: posAcomInit: using default phi matrix!!!");

/* copy ftsId */
ftsId = desired_ftsId;

/* set sample period */
Ts = fabs(trajGenPeriod);

#if 1
/*-----
  print out system order, frequency, damping, and sample period
*/
logMsg("\n\n");

sprintf(buffer1, "%5d %5d %5d %5d %5d %5d",

```

```

        system_order[0], system_order[1], system_order[2],
        system_order[3], system_order[4], system_order[5] );
logMsg(" P.A. Sys Order: %s\n", buffer1);

sprintf(buffer2, "%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
        w_n.v[0], w_n.v[1], w_n.v[2],
        w_n.v[3], w_n.v[4], w_n.v[5] );
logMsg(" P.A. Nat Freq : %s\n", buffer2);

sprintf(buffer3, "%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
        zeta.v[0], zeta.v[1], zeta.v[2],
        zeta.v[3], zeta.v[4], zeta.v[5] );
logMsg(" P.A. Zeta      : %s\n", buffer3);

logMsg(" P.A. Period  : %.4f\n", Ts);

logMsg("\n\n");
#endif

return(status);

} /*** end of function: posAcomInit ****/

/**** function: getForceTorque *****/
|
| Description: This is an access function for the measured force&torque.
|
|
|---rev---date-----author-----description-----
| 0.1   10/11/91  MJ Ryan    initial release
| 0.2   10/21/91  MJ Ryan    added call to spatPhiTransMult
| 0.3   10/23/91  MJ Ryan    added call to ftsLib
| 0.4   10/25/91  MJ Ryan    Use F_T_Scale vector now
| 0.5   10/28/91  MJ Ryan    F_T_Scale vector now fixed inside here
| 0.6   11/11/91  MJ Ryan    Removed F_T_Dead_Band; implemented elsewhere
*****/
static PAC_STATUS_TYPE          /* ret: void          */
getForceTorque
( VECTOR6 *toolTip_F_T ) /* out: tool tip forces */
{
    const VECTOR6
        F_T_Scale = { F_CONV, F_CONV, F_CONV, T_CONV, T_CONV, T_CONV };

    PAC_STATUS_TYPE
        status = OK;

    int
        i,

```

```

ftsStatus;

short int
temp_F_T[6] = {0, 0, 0, 0, 0, 0};

VECTORS
measured_F_T = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

#if 0 /* to be used when ready */
/* get force-torque vector from chanLib */

if (chanftsvectorread(measured_f_t.v, slot, mode) != chan_fts_ok )
{
logMsg("error using chanftsvectorread ");
measured_F_T = Zero_6;
}

/*? temporary fix before force drivers are ready */
measured_F_T = Zero_6;
#endif

/* call force sensor directly */
ftsStatus = ftsRead( ftsId, temp_F_T );
if (ftsStatus != OK)
{
logMsg("error using ftsRead \n");
if (ftsStatus = FTS_SENSOR_OVERLOAD)
return(PAC_FTS_OVERLOAD);
else
return(PAC_FTS_ERROR);
}

/* Convert forces&torques into mks units */
for (i=0; i<6; i++)
{
measured_F_T.v[i] = (float) (temp_F_T[i]) ;
measured_F_T.v[i] *= F_T_Scale.v[i];
}

/*-----
premultiply force vector by Phi matrix to translate forces & torques
into the the tool-tip frame.
-----*/
if ( spatPhiTransMult(toolTip_F_T,
&measured_F_T,
&toolTipToSensorFrame ) == NULL )

```

```

{
    logMsg("error using spatPhiTransMult ");
    (*toolTip_F_T) = Zero_6;
    status = PAC_SPAT_ERROR;
}

/* if status is bad, always return 0 */
if (status != OK)
    (*toolTip_F_T) = Zero_6;

return(status);

}/** end of function: getForceTorque *****/

/** function: posAcomPathMod *****/
|
| Description: This function returns a delta-position transform given a
|              desired force vector. The desired impedance is set
|              using the posAcomInit function.
|
|              This function simulates six decoupled mass-spring-damper
|              systems. The proper state equations are chosen for system
|              orders of:
|                  0: spring only
|                  1: spring and damper
|                  2: mass, spring and damper
|
|-----rev----date-----author-----description-----
| 0.1  10/11/91  MJ Ryan    initial release
| 0.2  10/21/91  MJ Ryan    moved delta_rate & delta_pos into body.
| 0.3  10/28/91  MJ Ryan    Added Force & torque limits check, added status
| 0.4  10/28/91  MJ Ryan    Removed Force & torque limits check
| 0.5  10/29/91  MJ Ryan    Fixed spatRot calls: Swapped rot matrices
| 0.6  11/05/91  MJ Ryan    use spatPhiMult instead of spatRot now
| 0.7  11/08/91  MJ Ryan    Disable R matrix; Added rotation of linear
|                          delta's(x,y,z) with the rotation portion of the
|                          previous delta_trans.
| 0.8  11/11/91  MJ Ryan    Added F_T_dead_zone processing
| 0.9  04/23/92  MJ Ryan    Fixed delay of 1; cleanup
|
|-----/
PAC_STATUS_TYPE          /*ret: status          */
posAcomPathMod
( VECTOR6 *desired_F_T, /* in: desired force/torque vector: tool-frame */
  TRANSFORM *delta_trans, /*out: delta-transform          */
  VECTOR6 *actual_F_T ) /*out: actual force/torque: tool-frame */
{
    PAC_STATUS_TYPE
    status = OK; /* init status to OK = 0 */

```

```

int
    i,
    temp;

static int
    k      = 0, /* current state, k */
    k_1    = 1; /* previous state, k-1 */

/* use this enum with VECTOR6 types */
enum
    { lx, ly, lz, rx, ry, rz};

VECTOR6
    delta_accel, /* acceleration variables in compliance frame */
    delta_vector, /* output delta vector in tool-tip frame */
    toolTip_F_T, /* measured forces&torque in tool-tip frame */
    sumTool_F_T; /* sum of force&torques in tool-tip frame */

VECTOR3
    deltaLin      = { 0.0, 0.0, 0.0 },
    deltaLinRotated = { 0.0, 0.0, 0.0 };

/* check for null pointers */
if (desired_F_T == NULL || delta_trans == NULL || actual_F_T == NULL)
{
    logMsg
        ("\007\007\007 WARNING!!!: posAcomPathMod: NULL pointers passed!!!");
    return (PAC_BAD_POINTERS);
}

/* get the current tool-tip forces and torques */
status = getForceTorque( &toolTip_F_T );
if (status != OK)
    return(status);

/* Copy tool-tip f/t into acutal_F_T for feedback */
(*actual_F_T) = toolTip_F_T;

for (i=0; i<6; i++)
{
    /* sum forces/torques on/around tool-tip; use select vector */
    sumTool_F_T.v[i]
        = ( toolTip_F_T.v[i] + desired_F_T->v[i] ) * select_vector.v[i] ;

    /* implement F_T_dead_zone */
    if ( fabs(sumTool_F_T.v[i]) <= F_T_dead_zone.v[i])
        sumTool_F_T.v[i] = 0.0;
}

```

```

else
  if ( sumTool_F_T.v[i] >= 0.0 )
    sumTool_F_T.v[i] -= F_T_dead_zone.v[i] ;
  else
    sumTool_F_T.v[i] += F_T_dead_zone.v[i] ;
}

/*-----
  Evaluate Mass-spring-damping state equations (10/11/91 MJR)
-----*/
for (i=0; i<6; i++)
{
  switch(system_order[i])
  {
  case 2:
    delta_accel.v[i]
      = ( sumTool_F_T.v[i]
          - impedance.damping.v[i] * delta_rate[k_1].v[i]
          - impedance.spring.v[i] * delta_pos[k_1].v[i] )
        / impedance.mass.v[i] ;

    delta_rate[k].v[i]
      = (delta_accel.v[i] * Ts) + delta_rate[k_1].v[i];

    delta_pos[k].v[i]
      = (delta_rate[k].v[i] * Ts) + delta_pos[k_1].v[i];
    break;

  case 1:
    delta_rate[k].v[i]
      = ( sumTool_F_T.v[i]
          - impedance.spring.v[i] * delta_pos[k_1].v[i] )
        / impedance.damping.v[i] ;

    delta_pos[k].v[i]
      = (delta_rate[k].v[i] * Ts) + delta_pos[k_1].v[i] ;
    break;

  case 0:
    delta_pos[k].v[i]
      = ( sumTool_F_T.v[i] ) / impedance.spring.v[i] ;
    break;

  }/* end switch */

}/* end for */

/* copy deltas into local variable for manipulation */
delta_vector = delta_pos[k];

/* Shift k states into k-1 states  by swapping k and k_1 indecies */

```

```

temp = k_1;
k_1 = k;
k = temp;

/*----- end of mass-damper-spring equations -----*/

/* extract linear delta into VECTOR3 type */
deltaLin.x = delta_vector.v[lx];
deltaLin.y = delta_vector.v[ly];
deltaLin.z = delta_vector.v[lz];

/* multiply linear deltas here using transVecPostMult
with oldRotationTrans */
if ( transVectPostMult( &deltaLinRotated,
                        &oldRotationTrans,
                        &deltaLin          ) == NULL )
{
    logMsg("\007\007\007 Error using transVectorPostMult ");
    /* with error use linear deltas */
    deltaLinRotated = deltaLin;
    status = PAC_TRANS_ERROR;
}

/* copy rotated linear deltas back into delta vector */
delta_vector.v[lx] = deltaLinRotated.x;
delta_vector.v[ly] = deltaLinRotated.y;
delta_vector.v[lz] = deltaLinRotated.z;

/* convert six vector of XYZRPY into 4x4 transform */
if ( spatToTransform( delta_trans,
                     &delta_vector,
                     FASTX          ) == NULL )
{
    logMsg("Error using spatToTransform ");
    /* with error use identity trans */
    delta_trans = transIdentityMake( delta_trans, FASTX);
    status = PAC_TRANS_ERROR;
}

/* find next oldRotationTrans using transConvert */
oldRotationTrans = *delta_trans;
if ( transConvert( &oldRotationTrans,
                  ROT          ) == NULL )
{
    logMsg("Error using transConvert ");
    /* with error use identity transform */
    oldRotationTrans = identityTrans;
    status = PAC_TRANS_ERROR;
}

```

```
    return(status);  
}/** end of function: posAcomPathMod *****/  
  
/** end of file: pathModLib.c *****/
```

```

/* %% %G% */

/*-
**          NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**          1991 ALL RIGHTS RESERVED.
**
**
**
** Permission to use, distribute, and copy is granted ONLY for research
** purposes, provided that this notice is displayed and the author is
** acknowledged.
**
** This software was developed at the facilities of the Center for
** Intelligent Robotic Systems for Space Exploration, Troy, New York,
** thanks to generous project funding by NASA.
**
*/

/*
** File:      compParams.h
** Written by: MJ Ryan

** Purpose:   This file holds declarations used by compParams.c

** Modification History:
** 0.1 11/11/91 MJ Ryan initial release

*/

#ifndef INCcompParamsh
#define INCcompParamsh

#include "pathModLib.h"

/* force/torque compliance parameters */
typedef struct
{
    IMPEDANCE_TYPE
    impedance;          /* the desired impedance structure */
    VECTORS
    select_vector,     /* six vector selecting with axis to comply in */
    desired_force,     /* six vector of desired forces */
    force_dead_zone;   /* dead zone of summed forces */
    int
    force_threshold_enabled; /* boolean to turn force threshold on */
    VECTORS
    force_threshold_percent, /* +/- % of desired_force that actual_force */
                                /* must attain before insertion is deemed */
}

```

```
                /* "complete" */
    force_threshold_time; /* time (secs) actual force must be in threshold */
} COMP_PARAMS_TYPE;

#define COMP_FILE_VERSION (2)

#define COMP_FILE "/home/mryan/mcs/installed/pathModLib/compParams.dat"

typedef enum
{
    BAD_COMP_PARAMS_FILE = 1
} COMP_STATUS_TYPE;

extern COMP_STATUS_TYPE
compParamsRead( const char    *file,
                COMP_PARAMS_TYPE *comp );

extern void
compParamsList( COMP_PARAMS_TYPE *comp );

#endif INCcompParamsh

/***** End of compParams.h *****/
```

```
/* %% %G% */
```

```
/*-
```

```
**          NOTICE OF COPYRIGHT
```

```
**          Copyright (C) Rensselaer Polytechnic Institute.
```

```
**          1991 ALL RIGHTS RESERVED.
```

```
**
```

```
**
```

```
**
```

```
** Permission to use, distribute, and copy is granted ONLY for research  
** purposes, provided that this notice is displayed and the author is  
** acknowledged.
```

```
**
```

```
** This software was developed at the facilities of the Center for  
** Intelligent Robotic Systems for Space Exploration, Troy, New York,  
** thanks to generous project funding by NASA.
```

```
**
```

```
*/
```

```
/*
```

```
** File:      compParams.c
```

```
** Written by: MJ Ryan
```

```
** Purpose:   This file holds functions for reading, listing, etc. the  
              compliance parameters used in the pathModLib functions
```

```
** Modification History:
```

```
** 0.1 11/11/91 MJ Ryan initial release
```

```
*/
```

```
#include "msgLib.h"  
#include "vxWorks.h"  
#include "stdioLib.h"  
#include "logLib.h"  
#include "cirsse.h"  
#include "mcsLib.h"  
#include "transLib.h"
```

```
#include "compParams.h"
```

```
#define MAX_LINE_SIZE 80
```

```
/******
```

```
** Routine:   compParamsRead()
```

```
** Purpose:   This routine reads the position accommodation parameters
```

```

**          from a file.
** Mod:
** 0.1 10/25/91 MJ Ryan Added sensor-gains to Comp struct.
** 0.2 10/25/91 MJ Ryan Removed sensor-gains to Comp Struct.
** 0.3 11/11/91 MJ Ryan Added additional parameters.

*/

COMP_STATUS_TYPE          /*ret: status */
compParamsRead( const char   *file,      /* in: comp-params file name */
                COMP_PARAMS_TYPE *comp ) /*out: comp-params structure */
{
    FILE *dataFile;
    char line[MAX_LINE_SIZE + 1];
    int
        compParamsVersion = 0;

    /* use default tape file if none specified */
    if (file == NULL)
        file = COMP_FILE;

    printf("\n Reading file %s ...",file);

    /* open file */
    if ((dataFile = fopen(file, "r")) == NULL)
    {
        fclose(dataFile);
        printf("\n Could not open file!\n");
        return (BAD_COMP_PARAMS_FILE);
    } /* end of if */

    /* skip comments */
    do
    {
        fgets(line, MAX_LINE_SIZE, dataFile);
    }
    while ((line[0] == '%') || (line[0] == '#'));

    /* get compParams.dat version number */
    sscanf(line,"%d",&compParamsVersion);

    /* check version */
    if (compParamsVersion != COMP_FILE_VERSION)
    {
        fclose(dataFile);
        logMsg("\n\007 \007 \007 Wrong compParams.dat version!!! ");
        return (BAD_COMP_PARAMS_FILE);
    } /* end of if */
}

```

```

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get select vector */
sscanf(line, "%f %f %f %f %f %f",
        &(comp->select_vector.v[0]), &(comp->select_vector.v[1]),
        &(comp->select_vector.v[2]), &(comp->select_vector.v[3]),
        &(comp->select_vector.v[4]), &(comp->select_vector.v[5]));

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get impedance mass/inertia vector */
sscanf(line, "%f %f %f %f %f %f",
        &(comp->impedance.mass.v[0]), &(comp->impedance.mass.v[1]),
        &(comp->impedance.mass.v[2]), &(comp->impedance.mass.v[3]),
        &(comp->impedance.mass.v[4]), &(comp->impedance.mass.v[5]));

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get impedance damping vector */
sscanf(line, "%f %f %f %f %f %f",
        &(comp->impedance.damping.v[0]), &(comp->impedance.damping.v[1]),
        &(comp->impedance.damping.v[2]), &(comp->impedance.damping.v[3]),
        &(comp->impedance.damping.v[4]), &(comp->impedance.damping.v[5]));

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get impedance spring vector */
sscanf(line, "%f %f %f %f %f %f",
        &(comp->impedance.spring.v[0]), &(comp->impedance.spring.v[1]),
        &(comp->impedance.spring.v[2]), &(comp->impedance.spring.v[3]),
        &(comp->impedance.spring.v[4]), &(comp->impedance.spring.v[5]));

```

```

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get desired force vector */
sscanf(line,"%f %f %f %f %f %f",
        &(comp->desired_force.v[0]), &(comp->desired_force.v[1]),
        &(comp->desired_force.v[2]), &(comp->desired_force.v[3]),
        &(comp->desired_force.v[4]), &(comp->desired_force.v[5]));

/*----- mjr 11/11/91 -----*/

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get force_dead_zone vector */
sscanf(line,"%f %f %f %f %f %f",
        &(comp->force_dead_zone.v[0]), &(comp->force_dead_zone.v[1]),
        &(comp->force_dead_zone.v[2]), &(comp->force_dead_zone.v[3]),
        &(comp->force_dead_zone.v[4]), &(comp->force_dead_zone.v[5]));

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get force_threshold_enabled bool */
sscanf(line,"%d", &(comp->force_threshold_enabled) );

/* skip comments */
do
{
    fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get threshold_percent vector */
sscanf(line,"%f %f %f %f %f %f",
        &(comp->force_threshold_percent.v[0]),
        &(comp->force_threshold_percent.v[1]),

```

```

&(comp->force_threshold_percent.v[2]),
&(comp->force_threshold_percent.v[3]),
&(comp->force_threshold_percent.v[4]),
&(comp->force_threshold_percent.v[5]));

/* skip comments */
do
{
  fgets(line, MAX_LINE_SIZE, dataFile);
}
while ((line[0] == '%') || (line[0] == '#'));

/* get threshold_time vector */
sscanf(line,"%f %f %f %f %f %f",
        &(comp->force_threshold_time.v[0]), &(comp->force_threshold_time.v[1]),
        &(comp->force_threshold_time.v[2]), &(comp->force_threshold_time.v[3]),
        &(comp->force_threshold_time.v[4]), &(comp->force_threshold_time.v[5]));

/* close file */
fclose(dataFile);

/* print out parameters */
compParamsList(comp);

logMsg("\n Done!\n");
return (OK);

} /* end of compParamsRead() */

/*****
** Routine:    compParamsList()
** Parameters: none.
** Returns:   none.
** Purpose:   Displays all information stored in the tgen comp
**           structure. This routine is used for debugging purposes.
** MOD
** 11/11/91 MJ Ryan Added additional params to print out
**/
void compParamsList( COMP_PARAMS_TYPE *comp )
{
  char buff1[80];

  /* display data */
  logMsg("\n\n");
  sprintf(buff1,"%f %f %f %f %f %f\n",
          comp->select_vector.v[0], comp->select_vector.v[1],

```

```

    comp->select_vector.v[2], comp->select_vector.v[3],
    comp->select_vector.v[4], comp->select_vector.v[5]);
logMsg(" Select : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->impedance.mass.v[0], comp->impedance.mass.v[1],
    comp->impedance.mass.v[2], comp->impedance.mass.v[3],
    comp->impedance.mass.v[4], comp->impedance.mass.v[5]);
logMsg(" Mass : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->impedance.damping.v[0], comp->impedance.damping.v[1],
    comp->impedance.damping.v[2], comp->impedance.damping.v[3],
    comp->impedance.damping.v[4], comp->impedance.damping.v[5]);
logMsg(" Damping : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->impedance.spring.v[0], comp->impedance.spring.v[1],
    comp->impedance.spring.v[2], comp->impedance.spring.v[3],
    comp->impedance.spring.v[4], comp->impedance.spring.v[5]);
logMsg(" Spring : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->desired_force.v[0], comp->desired_force.v[1],
    comp->desired_force.v[2], comp->desired_force.v[3],
    comp->desired_force.v[4], comp->desired_force.v[5]);
logMsg(" Des F/T : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->force_dead_zone.v[0], comp->force_dead_zone.v[1],
    comp->force_dead_zone.v[2], comp->force_dead_zone.v[3],
    comp->force_dead_zone.v[4], comp->force_dead_zone.v[5]);
logMsg("Dead-Zone : %s",buff1);

if (comp->force_threshold_enabled)
    sprintf(buff1, "Force-Threshold-Enabled = TRUE\n");
else
    sprintf(buff1, "Force-Threshold-Enabled = FALSE\n");
logMsg ("%s", buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->force_threshold_percent.v[0], comp->force_threshold_percent.v[1],
    comp->force_threshold_percent.v[2], comp->force_threshold_percent.v[3],
    comp->force_threshold_percent.v[4], comp->force_threshold_percent.v[5]);
logMsg("Threshld %% : %s",buff1);

sprintf(buff1,"% .3f % .3f % .3f % .3f % .3f % .3f\n",
    comp->force_threshold_time.v[0], comp->force_threshold_time.v[1],
    comp->force_threshold_time.v[2], comp->force_threshold_time.v[3],
    comp->force_threshold_time.v[4], comp->force_threshold_time.v[5]);
logMsg("Threshld Time: %s",buff1);

```

```
    logMsg("\n\n");  
  } /* end of tgenCompList() */  
  
/***** end of file compParams.c *****/
```

