A COMPARISON OF FORCE CONTROL ALGORITHMS FOR ROBOTS IN CONTACT WITH FLEXIBLE ENVIRONMENTS

by

Lee S. Wilfinger

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

December 1992

CIRSSE REPORT #135

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In order to perform useful tasks, the robot end-effector must come into contact with its environment. For such tasks, force feedback is frequently used to control the interaction forces. Control of these forces is complicated by the fact that the flexibility of the environment affects the stability of the force control algorithm. Because of the wide variety of different materials present in everyday environments, it is necessary to gain an understanding of how environmental flexibility affects the stability of force control algorithms.

This report presents the theory and experimental results of two force control algorithms: Position Accommodation Control and Direct Force Servoing. The implementation of each of these algorithms on a two-arm robotic testbed located in the Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) is discussed in detail. The behaviour of each algorithm when contacting materials of different flexibility is experimentally determined. In addition, several robustness improvements to the Direct Force Servoing algorithm are suggested and experimentally verified. Finally, a qualitative comparison of the force control algorithms is provided, along with a description of a general tuning process for each control method.

CHAPTER 1
Introduction

Force control (or compliance control) is the control of the forces which are exerted by a robot on the environment. These forces are applied via the end–effector of the robot, and are ultimately controlled by controlling the motor torques of the robot joints. Force control algorithms can be roughly divided into two categories:

- direct methods
- indirect (impedance–based) methods

Direct methods control forces exerted by the robot by controlling joint torques. The forces measured at the end–effector are used to directly calculate joint torques by using the transpose Jacobian of the robot arm. The control law often involves a proportional or integral term. as well as a feedforward term. Throughout this document, this type of control will be referred to as Direct Force Servoing (DFS).

Indirect methods control forces by controlling end–effector positions. Force errors are converted to position errors, allowing a position–based control loop to drive the robot joints. Conversion of the force error into a position error is often done using a mass–spring–damper relationship; the resulting system causes the end–effector to respond to forces in a manner similar to a mass–spring–damper system.

Two examples of indirect algorithms are Position Accommodation Control (PAC) and Impedance Control. In Position Accommodation, the impedance relationship is commanded by providing the appropriate position setpoints to a joint controller. The position setpoints are generated based on the force error, according to a mass–spring–damper equation.

Impedance control produces similar end–effector behaviour, but is more complicated. Feedback linearization is used to cancel out the nonlinear robot dynamics.

As in the PAC algorithm, the end-effector responds to force errors according to a mass-spring-damper relationship.

1.1 Motivation for Force Control

In order to perform most useful tasks, a robot must come into contact with its environment. Many of these tasks require that the interaction forces be held within some range. Examples of such tasks include the handling of fragile payloads, the insertion and removal of parts, and deburring and machining.

In an ideal environment, the objects in the environment would be modeled precisely, and the robot end-effector could be positioned exactly. In such a situation, the forces on the environment could be controlled by sensing and controlling the position of the end-effector.

In a typical environment however, the location of objects is not known precisely. There may also be unknown objects in the robot workspace. In addition, the positioning accuracy of real robots is not perfect. Without compliance control, the robot may damage itself and its environment when it comes into contact with objects in its workspace. For this reason, force control is a necessary part of a robot control system.

In addition to allowing the robot to safely contact the environment, the force control algorithm should permit the robot to apply some desired force to the object being worked on. These tasks are complicated by the fact that different objects have varying degrees of flexibility, which will affect the stability of the force controller. Thus, a force control algorithm tuned for one type of surface may not work well when the robot encounters a different surface.

There are a myriad of different materials which exist in everyday environments; wood, cloth, rubber, glass, cardboard, and plastic are only a few examples. Each of these materials will react to a force differently. For a given force, each material will deform to a different degree. Further, when the applied force is removed, each material will return to its original shape at different rates. In order to operate safely within its environment, the robot should be able to come into contact with a wide variety of different materials.

The motivation for this report stems from this basic problem of allowing a robot to safely contact objects in an uncertain environment. It is necessary to gain an understanding of how environmental flexibility affects the behaviour and stability of force control algorithms. In order to acquire this knowledge, a number of different experiments were conducted using both the Position Accommodation Control and Direct Force Servoing algorithms.

The equations of the basic PAC and DFS algorithms were determined and examined in order to understand the behaviour of the algorithms. To confirm the theory, both control algorithms were implemented on the CIRSSE two-arm robotic testbed. A number of experiments were then conducted to verify the performance of the basic force control algorithms. In order to determine the effect of environmental flexibility on the behaviour of each control algorithm, the robot was commanded to contact a variety of different surfaces, and apply a constant force.

In order to improve the performance of the DFS controller, the basic algorithm was augmented to make it more robust with respect to disturbances and impact forces. These modifications were then experimentally verified by applying a disturbance to the robot while it was pushing on its environment with a constant force.

1.2 Literature Review

Many force control algorithms have been proposed in recent years. A small sample of the vast amount of research being done in this field is listed below. Whitney [1] provides an interesting historical perspective on the force control problem, and outlines a number of different approaches that have been tried.

Hogan [2] presents the theory of impedance control. He also discusses issues related to the implementation of this control method.

Raibert and Craig [3] implemented a direct force controller on a modified Scheinman robot arm as part of a hybrid position/force control system. The feedback control law that was used was a PI–type. In contrast to most of the literature, their control law converted force errors from Cartesian space to joint space before applying the PI control law.

Khatib and Burdick [4] implemented direct proportional force control on a PUMA 560 arm equipped with a wrist force sensor. An active damping term was also included in the control law in order to improve the stability of the system.

An and Hollerbach [5] implemented a direct proportional force control algorithm on a three–link direct drive arm, in which the third link was purely force–controlled. In addition, they performed experiments in which the robot came in contact with aluminum and hard rubber surfaces. They noted that the algorithm had problems with stiffer surfaces like aluminum, but worked better with more compliant surfaces. They advocated putting a low–pass filter in the forward control loop to improve the stability properties of the controller.

Wedel and Saridis [6] discuss the implementation of a direct proportional force controller on a PUMA 560 arm with a 6 Degree Of Freedom (DOF) force sensor. Their work was based heavily on the hybrid force control algorithm proposed by Raibert and Craig, although the proportional control law operated on the Cartesian force error before converting to joint space. They noted significant steady state error in the force measurements. They also noted that the control algorithm tended to be unstable when the robot contacted stiff surfaces.

Wen and Murphy [7, 8] have looked at both Direct Force Servoing and Position Accommodation Control from a theoretical point of view. They have observed that direct integral force control is more robust than direct proportional force control with respect to time delay in the control loop. In addition, direct integral force control has better steady state error characteristics.

The observations made by Wen and Murphy have been confirmed by experimental results performed by Volpe [9]. Volpe and Khosla [10, 11] have implemented direct integral force control on the six-axis CMU DD Arm II. The control law that was used included an active damping term like that used in [4]. In addition to integral force control, Volpe and Khosla have experimented with direct proportional force control (with active damping) and impedance control.

1.3 Summary of Topics

The following topics are discussed in this report. Chapter 2 discusses the CIRSSE testbed, and the software which was used to perform all of the experiments described in this document. Chapters 3 and 4 discuss the theory and implementation of the Position Accommodation Control and Direct Force Servoing algorithms, respectively. Chapter 5 describes the experiments which were performed on the CIRSSE testbed, and discusses some of the results which were obtained. Chapter 6 recommends some modifications to the Direct Force Servoing algorithm, and presents experimental justification for the alterations. Chapter 7 compares the PAC and DFS algorithms on a number of issues, and also presents a general tuning process for each control algorithm. Finally, Chapter 8 summarizes the work presented here, and suggests avenues of future research.

Appendices A and B discuss the derivation of the gravity compensation and Jacobian equations, which form the basis for two real-time libraries used by the force control software. These libraries were essential to the implementation of both force control algorithms, and are discussed in Chapter 2.

Appendix C provides additional information on the robot joint controllers, which were implemented as part of the force control code presented in Chapter 2.

The research code outlined in Chapter 2 is not presented in this work, due its size. However, a copy of the research code, the collected data, and an electronic version of this document have been archived. Interested parties may contact the author at the Center for Intelligent Robotic Systems for Space Exploration, Rensselaer Polytechnic Institute, Troy, New York 12180.

1.4 Notation and Conventions

The following is a list of the notation and conventions used throughout this document:

- The term "end–effector" is used synonymously with "tool" and "gripper".
- Continuous functions of time are indicated with parentheses; for example: $f(t)$. Discrete–time functions are indicated using square brackets, as in: $f[k]$.
- The coordinate frames of the CIRSSE testbed arm are labeled 1 – 9; frame 0 is the global origin [12]. An E denotes the end–effector frame.
- Due to the design of the CIRSSE testbed arm, the joints of the PUMA (the PUMA is considered to be part of the full arm) are labeled 4 – 9. The subscript labels of parameters reflect this convention as well. Thus, link 6 of the PUMA is considered link 9 of the arm, and the mass of this link would be labeled m_9.
- Estimated values will be specified with hat symbols over them. Thus, an estimate of the acceleration due to gravity would be designated as $\hat{g}$, while the actual value would be specified as g.
- ${}^{k}p_{i,j}$ is the 3×1 vector describing the position of frame j with respect to frame i, expressed in the coordinates of frame k. Under many circumstances,

$k = i$. In this case, the i subscript will be dropped; thus, ${}^k p_{k,j} = {}^k p_j$. Note that ${}^k p_{i,j} = -{}^k p_{j,i}$.

- ${}^i_j R$ is the 3×3 rotation matrix describing the orientation of frame j with respect to frame i.

- ${}^i_j T$ is the 4×4 homogeneous transformation describing the position and orientation of frame j with respect to frame i. Thus:

$$ {}^i_j T = \begin{bmatrix} {}^i_j R & {}^i p_j \\ 0 & 1 \end{bmatrix} $$

  For a detailed discussion on the subject of homogeneous transforms, the reader is referred to [13].

- ${}^k \tilde{p}_{i,j}$ is the 3×3 cross product matrix associated with the vector ${}^k p_{i,j}$, expressed in the coordinates of frame k:

$$ {}^k \tilde{p}_{i,j} = \begin{bmatrix} 0 & -{}^k p_{i,j}(z) & {}^k p_{i,j}(y) \\ {}^k p_{i,j}(z) & 0 & -{}^k p_{i,j}(x) \\ -{}^k p_{i,j}(y) & {}^k p_{i,j}(x) & 0 \end{bmatrix} $$

  The arguments x, y, and z in the above matrix represent the three components of the vector ${}^k p_{i,j}$. The term "cross product matrix" indicates that for any 3×1 vector ${}^k w$, the following equation holds:

$$ {}^k \tilde{p}_{i,j} \, {}^k w = {}^k p_{i,j} \times {}^k w $$

  It should be noted that the following equation also holds for cross product matrices: ${}^l_k R \, {}^k \tilde{p}_{i,j} \, {}^k_l R = {}^l \tilde{p}_{i,j}$

- ${}^k \dot{x}_{i,j}$ is the velocity of frame j with respect to frame i, expressed in the coordinates of frame k. This is a 6×1 vector; the first three components of this vector are the linear velocity and the last three are the angular velocity:

$$ {}^{k}\dot{x}_{i,j} = \begin{bmatrix} {}^{k}\dot{x}_{L} \\ {}^{k}\omega \end{bmatrix} $$

Note that under many circumstances, $k = i$. In this case, the i subscript will be dropped; thus ${}^{k}\dot{x}_{k,j} = {}^{k}\dot{x}_{j}$.

CHAPTER 2
Description of Testbed

This chapter describes the hardware and software which comprise the CIRSSE testbed. The first section discusses the hardware used in the force control experiments. The second section discusses the software which controlled the experiments, how the software was organized, and the function of each major software module.

2.1 Hardware

This section discusses the testbed at CIRSSE upon which all force control experiments were performed. Figures 2.1 and 2.2 show a portion of the CIRSSE Sun–based computer network, and its connection to the CIRSSE testbed. The testbed consists of two 9 DOF robot arms. Each arm consists of a PUMA arm (either a PUMA 560 or PUMA 600), mounted on a 3 DOF platform. Each arm is equipped with a force sensor and specialized gripper, and is controlled by a VME cage with multiple processors.

All force control experiments discussed in this report were run on only one of the arms. For this reason, further description of the testbed only mentions the single arm (and its sensors) used in the experiments.

2.1.1 Sun Network

At CIRSSE, there are a number of Sun workstations which are connected together via an Ethernet link. The VME cage which controls the testbed is also connected to this network. This allows users to compile their code on the Suns, and then download the code to the cage for execution. Further, any data that is collected from experiments can be efficiently sent to the Suns for analysis and storage.

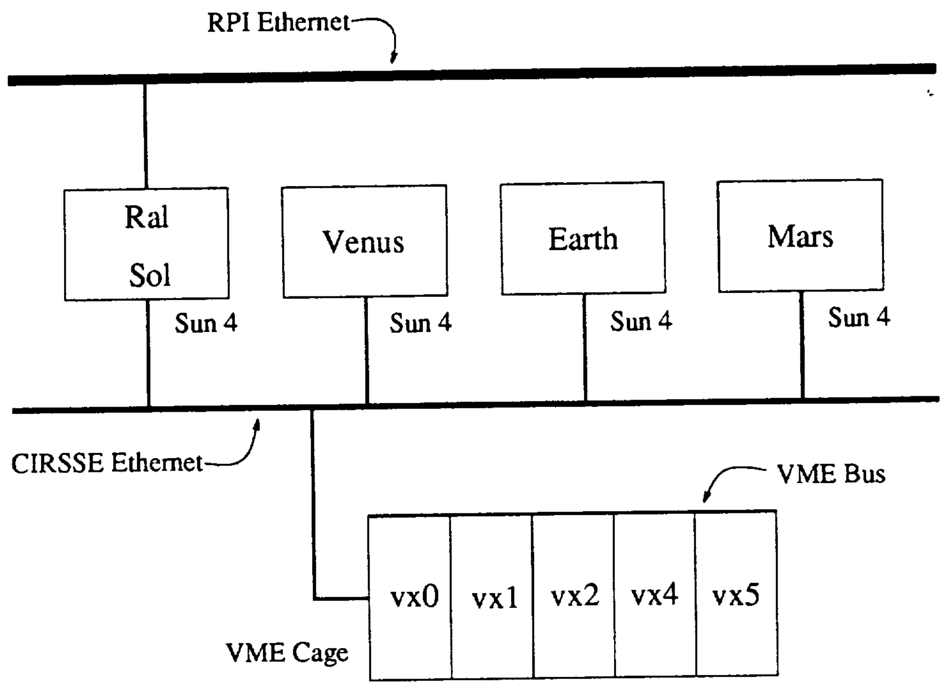


Figure 2.1: CIRSSE Computer Network and VME Cage

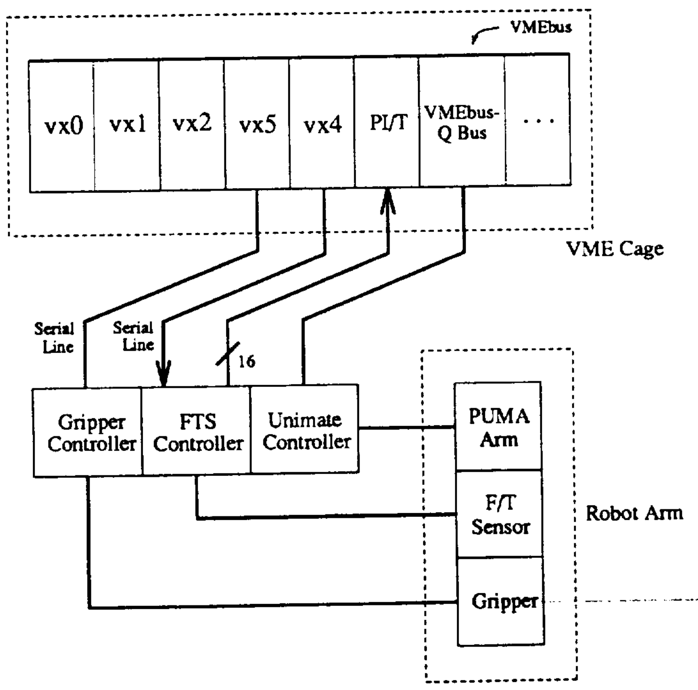


Figure 2.2: CIRSSE VME Cage and Connection to Testbed

2.1.2 VME cage

The VME cage currently contains 5 single-board processors; two MVME–135 and three MVME–147 boards. The processors run VxWorks, a real-time UNIX-based operating system, which provides a convenient environment for writing and testing real–time software. A more detailed description of the software used in this research is provided later in this chapter.

The cage also contains MVME–360A Parallel Interface/Timer (PI/T) boards for peripheral devices and sensors, as well as VMEbus–Q Bus adapter cards for communication with the Unimate controllers. In addition, the cage contains a 4–Megabyte shared memory card. This memory is accessible to all processors, and provides a convenient mechanism for tasks to communicate.

2.1.3 PUMA 600

The PUMA 600 is a 6 DOF industrial robot arm built by Unimation, Inc. Such robots are widely used for research at academic institutions, and represent commonly available technology.

PUMA arms are normally controlled using the VAL programming language. The controller units of the PUMAs at CIRSSE have been modified, however, so that VAL is not longer needed. Instead, a VMEbus–Q bus interface connects the Unimate controller to the VME cage, allowing the robots to be controlled by tasks running on the VME cage. In this configuration, the controller unit acts as a hardware interface and power supply for the PUMA.

2.1.4 Platform

Each PUMA is mounted on a 3 DOF platform to form a 9 DOF arm. See Figures 2.3 and 2.4. Due to its range of motion, this platform extends the workspace of the PUMA considerably. In addition, it allows users to conduct experiments in the area of redundant manipulators. For the research work presented here, the platform was used as a rigid base which could be easily positioned near the compliant surfaces to be tested. The platform was kept level for all of the force experiments, in order to help maintain a common robot configuration across all of the experiments that were performed.

2.1.5 Force Sensor

The PUMA 600 is equipped with a Lord 15/50 force sensor. This 6–axis sensor can read forces of up to 15 pounds and torques of up to 50 inch–pounds, and has a maximum sampling rate of 300 Hz. [14]. The force sensor is mounted rigidly on the flange of the PUMA arm. The force sensor controller unit was interfaced to the VME cage via an MVME-360A PI/T board. During normal operation, the force sensor transmits strain gauge signals to the VME cage. A driver routine then converts these numbers into forces and torques felt at the force sensor.

2.1.6 Gripper and Load

An aluminum end–effector is attached rigidly to the end of the force sensor on the robot arm. This gripper is electronically controlled and uses pneumatics to open and close the fingers. During force experiments, the gripper was required to hold a small object and apply force to the environment through this object.

2.1.7 Support Stand

All compliant surfaces were supported by a rectangular steel table (represented in Figures 2.3 and 2.4 as a box). The table brought the materials to be tested up to a height which was well inside the PUMA workspace, and thus easily reachable by the PUMA.

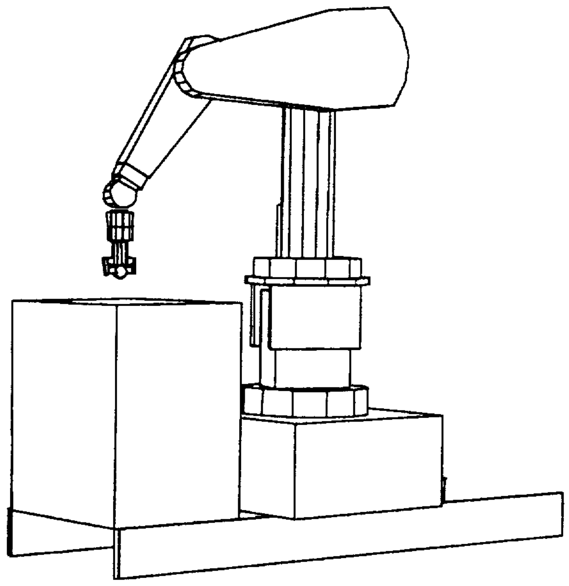


Figure 2.3: View of Test Setup for Force Control Experiments

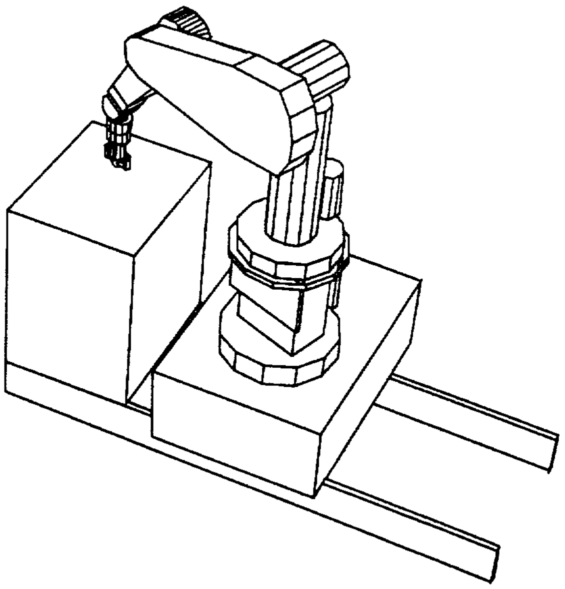


Figure 2.4: Second View of Test Setup for Force Control Experiments

2.2 Software

This section details the software that was written and used to conduct force control experiments on the testbed hardware. The software written for this report is quite extensive. Furthermore, the software is supported by a number of additional layers of code.

The software is discussed in three parts. First, the software environment in which the force control code executes is discussed. Next, the overall hierarchy of the force control code is described. Finally, the code is discussed in depth, from the code that was allocated to specific processors to a brief description of each major code library that was used for the experiments.

2.2.1 Software Environment

All code written to run on the CIRSSE testbed is actually supported by two layers of code, VxWorks and CTOS. These layers are described in the next section.

2.2.1.1 VxWorks

All of the processors in the VME cage run VxWorks. VxWorks is a UNIX–based real–time operating system produced by Wind River Systems, Inc. This operating system is relatively easy for programmers to use, because it has several important features [15, 16]:

- *Close network compatibility with UNIX* – this allows processes running under UNIX and VxWorks to communicate with each other efficiently.
- *Extensive run time libraries* – these include string and math libraries, linked list manipulation commands, and others. This provides a good foundation for code being written to execute under VxWorks; the very low-level code has already been written for the programmer.

- *Object code compatibility with UNIX* – this provides programmers with a very convenient development feature. Code can be written and compiled on the Sun workstations using standard UNIX tools (C compilers, assemblers, editors, etc.), The object code is then downloaded to the cage for execution.

- *Dynamic linking of object modules at load time* – the object modules are linked automatically upon downloading. No code needs to be linked ahead of time under the UNIX environment, since all linking is done on the VxWorks side at load time.

- *An interactive shell for debugging and development* – this allows users to interactively test and execute the code that is downloaded onto the VME cage processors. Once an object module is downloaded, the user has immediate access to all functions which are contained in the module. The functions can be called by name at the shell prompt.

All of these features of VxWorks give programmers a convenient, flexible environment for writing, testing, and running real–time code.

2.2.1.2 CTOS

VxWorks provides an excellent foundation for writing real–time code to control the CIRSSE testbed. However, there are some features that were needed which VxWorks lacks. For example, there is no standard way for tasks to communicate between processors. In order to supply some of the missing features, a body of code called CTOS was written.

CTOS is an acronym for "CIRSSE Testbed Operating System" and is the name of a layer of software which runs above the VxWorks real-time operating system, augmenting it [16]. See Figure 2.5. In addition, CTOS applications can run under UNIX, and communicate to processes on the VME cage. CTOS provides several

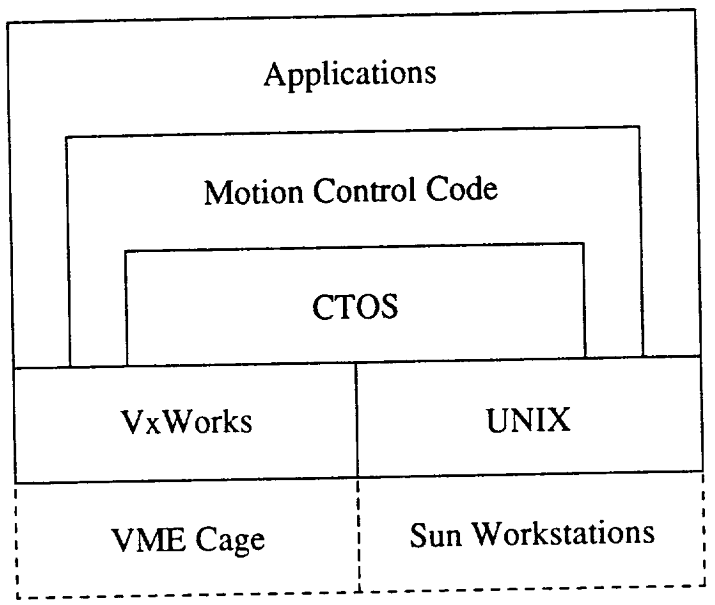


Figure 2.5: Location of CTOS Code Layer

features which VxWorks alone does not possess:

- *Inter-processor semaphores* – similar to conventional semaphores, these provide mutual exclusion for shared resources. Typically, conventional semaphores are implemented for efficient use by tasks executing on the same processor. However, semaphores which are used between processors are not implemented as efficiently, in general. For example, busy-waiting techniques might be employed.

  Inter-processor semaphores are designed to work efficiently between processors. Tasks on one processor can be blocked waiting for a resource being used by a task on a different processor.

- *Inter-processor blocks (IPBs)* – these are similar to semaphores, but function slightly differently. There is no ordering of the give and take operations on a semaphore; a task can give a semaphore before a second task takes it. In such a case, the second task is not blocked. With inter-processor blocks, a task which takes the block will always be blocked until it is released (the release must come *after* the block) by another task. Thus, there is a specific ordering which must be followed when using the IPBs.

- *Process synchronization across CPUs* – Individual processors can easily regulate their own processes. With the proper initialization, for example, a task might be unblocked to run every 5 milliseconds. A problem arises, however, when there are tasks on multiple processors that need to be synchronized (Figure 2.6A).

  CTOS provides a mechanism for ensuring that processes on different processors start at the same time, and remain synchronized. Therefore, tasks on different processors that have the same period of execution will be released "simultaneously" every period (Figure 2.6B).

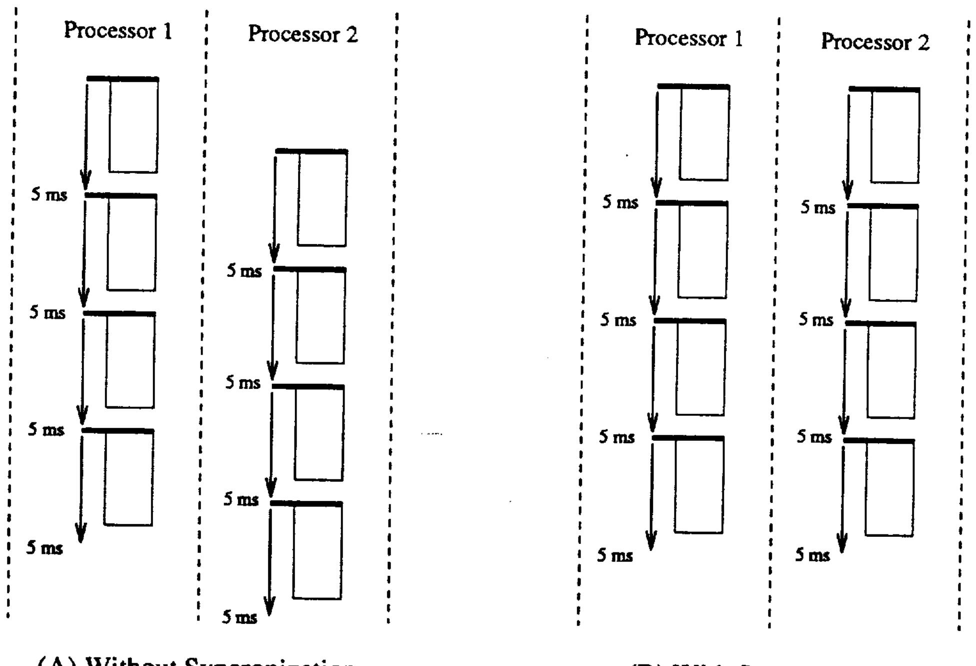


Figure 2.6: Effect of synchronization on task execution

- *Easy distribution of tasks on each CPU* – while booting the VME cage, CTOS requires a configuration file which lists the processes to be started and the processors to run the tasks on. It is a simple matter for the user to add or delete tasks from the list, or move tasks from one processor to another. The changes take effect when the VME cage is rebooted.

- *Message passing between tasks* – CTOS also supports message passing to applications running on the Sun workstations or the VME cage. This provides a valuable mechanism for communication between tasks on different processors. This allows tasks to transfer data conveniently from the VME cage to the workstations, where it can be stored for later analysis. It also allows the user interface to reside on the UNIX-based machines, so that standard Graphic User Interfaces (GUIs) such as X-Windows can be used.

2.2.2 Hierarchy of Force Control Code

This section describes each major component comprising the force control system, and the force control code used to conduct experiments.

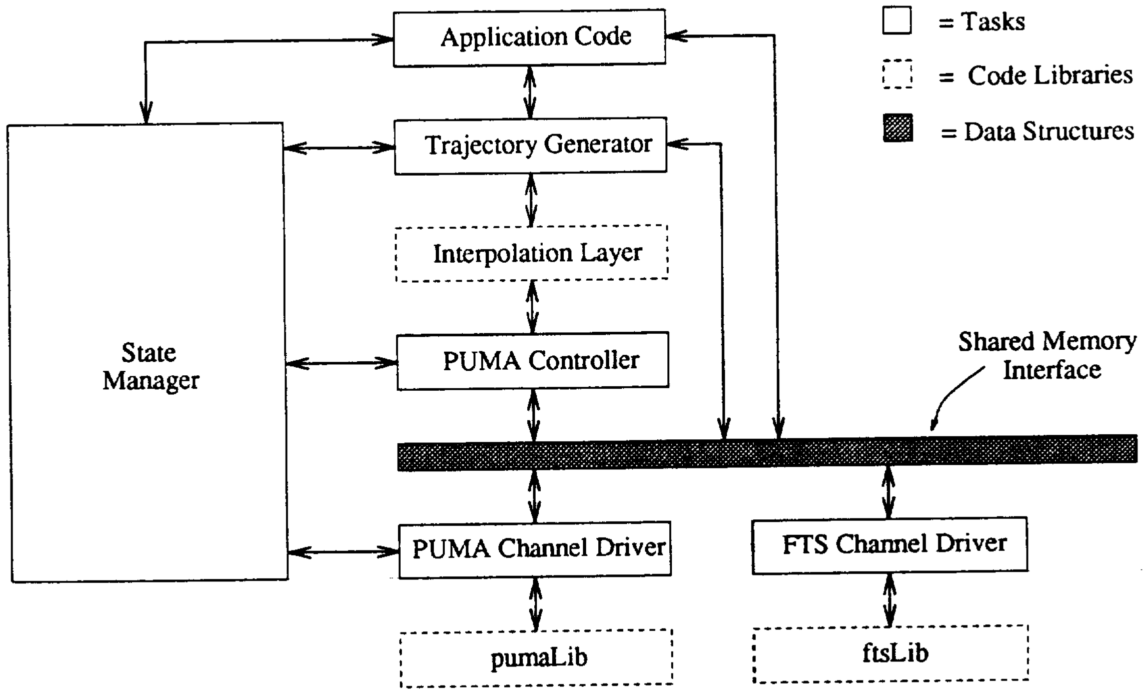


Figure 2.7: Relative Hierarchy of Major Code Components

Figure 2.7 shows the major components of the research code [16]. These include the application, trajectory generator, interpolator, joint controller, channel drivers, and state manager. The arrows in the figure indicate the communication paths between the pieces of code. Each part is described in more detail below.

2.2.2.1 Application

The application is a layer of code which provides the user interface. High level commands such as starting up and shutting down the testbed, and reading joint positions are available. In addition, commands are provided to allow the user to change motion control modes easily. Thus, the user has the ability to easily position the arm at a certain location and start a force control experiment.

In a typical robot control situation, the application would be the layer which would generate knotpoints for the trajectory generator to process. In this case, the user generates a knotpoint when the destination position of the robot is specified.

2.2.2.2 Trajectory Generator

The trajectory generator (TG) is the code layer which takes knotpoints from the application code, and generates more finely spaced setpoints for the joint controller. The TG used in the force experiments used simple, joint interpolated motion to move between the current position and a destination position specified by the application code.

In addition to the movement mode, where joint interpolated setpoints are generated, the TG also has three other modes; idle mode, PAC mode and DFS mode. Idle mode is the default mode, in which the TG simply maintains the current position of the robot arm. In order to switch the controller from one mode to another, it is necessary to pass through idle mode; in this way, the robot is assured of being motionless when the new motion mode is begun.

By requiring that the robot be motionless at the start of each motion mode, the trajectory generation code can be made simpler. This is because the code to perform one type of movement does not have to be able to handle transient motion resulting from previous motion modes.

In DFS mode, the TG monitors the position of the robot arm, but doesn't generate setpoints for the joint controller. In PAC mode, the TG executes an algorithm which implements the Position Accommodation force control. A description of the theory and implementation of PAC is located in Chapter 3.

2.2.2.3 Interpolator

The interpolator resides between the trajectory generator and the controller. This layer performs linear interpolation on the setpoints which the TG provides for the controller. This layer is necessary because of the fact that the TG typically runs much slower than the joint controller. For example, the joint controller might run every millisecond, while the TG may run every 20 milliseconds.

The interpolator obtains the sampling periods of the TG and joint controller upon system startup. From this information, it determines the correct number of interpolation points between the setpoints that the TG provides. For example, in the above scenario, the joint controller would require 20 interpolation points between the trajectory generator's setpoints, because the controller runs 20 times faster than the TG.

Because the interpolation layer is self-contained, and is accessed only by the joint controller and the trajectory generator, it is not noticed by the other modules in the system. Because the interpolator is so transparent, it is often convenient in discussions to ignore its presence altogether. However, in the implementation of a robotic control system, the interpolator is an important module which can not be ignored.

2.2.2.4 Joint Controller

The function of the controller is to take setpoints from the interpolator and apply a control torque to the robot joints, in order to servo the joints to the desired positions. As the setpoints are changed, the controller calculates the torques required to move the arm joints so that the joint positions track the setpoints.

It should be noted that the controllers calculate a torque for the motors of the PUMA arm. Because of the gear train of the PUMA, the torque seen from the link side of the gear train (hereafter referred to as "joint torque") is much higher than the torque seen from the motor side ("motor torque"). This is due to the gear ratio. The control torques which are calculated must take into account the gear ratios of the PUMA joints.

The controller which was used has several different modes of operation. In each mode, a different control equation is used to generate the motor torques. Messages can be sent from the application layer in order to have the controller switch modes. In order to keep the controller as simple as possible, the mode was switched only when the arm was stationary; this prevented the need for having to deal with any motion transients that might occur if the control mode was switched while the robot was moving.

The control modes supported by the controller are:

- *Proportional-Integral-Derivative (PID) mode*: This mode was used for to initially position the arm prior to running experiments. Because of the integral term, this mode exhibited the best positioning accuracy. The control equation that was used was:

$$\tau = \widehat{M}(\theta)[\ K_P(\theta_d - \theta) + K_I \int_0^t (\theta_d - \theta)dt + K_D(\dot{\theta}_d - \dot{\theta})\ ] + \widehat{g}(\theta)$$

In the above equation, $\widehat{g}(\theta)$ is the estimated torque needed in order to compensate for gravity. $\widehat{M}(\theta)$ is an estimate of the mass matrix of the arm links.

To simplify the calculations, $\widehat{M}(\theta)$ is assumed to be diagonal. containing the dominant configuration–dependent terms. The feedback control gains and the formula for the mass matrix which were used in the controller are contained in Appendix C.

In actuality, the trajectory generator only calculated position setpoints (all desired velocities were zero), so $\dot{\theta}_d = 0$ in the above equation. Thus, the controller really amounts to a PI controller with rate feedback [17].

- *Proportional-Derivative (PD) mode*: This mode was activated whenever the Position Accommodation Control was turned on. The control equation describing this mode is:

$$\tau = \widehat{M}(\theta)[\, K_P(\theta_d - \theta) + K_D(\dot{\theta}_d - \dot{\theta})\,] + \widehat{g}(\theta)$$

As with the PID mode, $\dot{\theta}_d = 0$. The feedback gains are listed in Appendix C.

- *Gravity Compensation Mode*: In this mode, the joint positions were read, and the torques needed to compensate for the forces on the arm links due to gravity were calculated. Appendix A discusses the derivation of the gravity compensation equations used in this mode. The control equation is:

$$\tau = \widehat{g}(\theta)$$

- *DFS mode*: In this mode, the forces on the end–effector are used in the calculation of the control torque. The basic control equation that was used is:

$$\tau = J^T F + \widehat{g}(\theta)$$

F is a control signal which is a function of the desired and actual forces being applied to the environment by the end–effector. This control mode is described in detail in Chapter 4.

2.2.2.5 PUMA Channel Driver

The PUMA channel driver communicates directly with the Unimate controller unit. It is one of the lowest level tasks in the system, and handles data I/O for the controller. The main function of the channel driver is to supply the Unimate controller with motor torques (calculated by the joint controllers) and to supply the joint controllers (and other tasks) with the current position of the joints. In order to communicate this information, the data is stored in shared memory locations which are accessible by the channel driver and the tasks that require joint information.

2.2.2.6 FTS Channel Driver

The force/torque sensor channel driver performs a function similar to that of the PUMA channel driver; it acts as the interface to the force sensor for all tasks running on the cage. During normal operation, the channel driver reads the force sensor periodically and copies the force sensor readings into a shared memory location. This location can then be accessed by all tasks which require force information.

Due to the startup requirements of the force sensor [14], the sensor is connected directly (via serial cable) to one processor on the VME cage. Because of this, the channel driver is constrained to reside on the same processor. However, by copying the force readings into shared memory, tasks which need the force information can be distributed across the other processors.

It is worth noting that the LORD 15/50 force sensor is a 300 Hz force sensor. Thus, the rate at which the channel driver can be run is limited by this value. This sampling rate also limits the gains that can be used in any force control algorithm.

2.2.2.7 State Manager

The State Manager acts as a coordinator between all processes, informing them of major changes in the state of the system. Upon booting the VME cage, the state manager will help coordinate tasks which are started, and will provide them with information (such as their sampling periods) that they need in order to properly execute their respective functions.

2.2.3 Further Description of Force Control Code

In the previous section, the major logical components of the force control system were described. In this section, the pieces of code which comprise the logical units are described.

The code that was written to conduct the force control experiments was split across all five processors in the VME cage. All code executed concurrently, communicating via message passing, inter–processor blocks and shared memory locations. Table 2.1 lists the pieces of code which were allocated to the various processors.

vx0 (MVME 147)	vx1 (MVME 135)	vx2 (MVME 135)	vx4 (MVME 147)	vx5 (MVME 147)
mcsLib	mcsLib	mcsLib	mcsLib	mcsLib
chanLib	chanLib	chanLib	chanLib	chanLib
interpLib	interpLib	interpLib	interpLib	interpLib
chanFtsLib	chanFtsLib	chanFtsLib	chanFtsLib	chanFtsLib
dataLog	dataLog	dataLog	dataLog	dataLog
smLib	pumaLib	transLib	ftsLib	transLib
gripper interface	chanPuma	spatLib	chanFtsDrv	spatLib
pacLib		kinLib		gravLib
transLib		pacLib		jacLib
spatLib		tgen		dfsLib
kinLib				ctrlPuma
dfsLib				gripper Drv
dataVxWorks				
application				

Table 2.1: Distribution of Processes on VME Cage

Note that the listings in the upper part of the table are common to all processors; thus, this code is identical across all processors. These pieces of code provide a common interface for all tasks being executed on the processor.

chanLib – This code contains routines for accessing the shared memory locations containing arm joint information (motor torques and joint positions).

chanFtsLib – This code contains routines for reading and writing force information to shared memory locations. Only the FTS channel driver writes forces to shared memory; all other tasks may read forces from shared memory.

dataLog – This code contains routines for logging data on the VME cage. Tasks which sample and record data will call functions in this piece of code.

interpLib – This code contains routines for working with the interpolation layer. The trajectory generator and joint controller call the functions located within this library.

mcsLib – This code contains routines which are called by tasks on all processors when the system is booted. It contains a number of functions for communicating with the State Manager.

The listings in the lower part of Table 2.1 are specific to each processor. Each piece of code will be described briefly here:

application – This is a part of the application layer shown in Figure 2.7. It provides the user with such commands as starting up and shutting down the CIRSSE testbed, moving the arm to specific locations, and reading the arm position. It also provides the user with commands to switch into and out of both PAC and DFS force control modes.

chanFtsDrv – This is the force/torque sensor channel driver.

chanPuma – This is the PUMA channel driver.

ctrlPuma – This is the joint controller for the PUMA arm.

dataVxWorks – This is another component of the application layer. It provides the user with commands for collecting ("logging") data on the VME cage, and uploading data to the Sun network.

dfsLib – Direct Force Servoing Library. This is a low-level library which provides the joint controller with functions for calculating the force control signal (F) for DFS control. It also provides file–reading functions which enable the application layer to read in new force control parameters from data files.

ftsLib – Force/Torque Sensor Library. This is the lowest level driver code for the force sensors in the lab. It provides the FTS channel driver (chanFtsDrv) with a number of functions to control the force sensor hardware easily.

gravLib – Gravity Compensation Library. This library contains routines for calculating the PUMA motor torques required to hold the PUMA arm stationary in the presence of gravity. The derivation of the equations used internally by gravLib are described in Appendix A, while a complete description of this code is given by [18].

gripper Drv – Gripper Driver. This is the lowest level code for the gripper mounted on the PUMA arm. The driver receives commands (via message passing) from the gripper interface code, and sends out the low level commands necessary to operate the gripper.

gripper interface – This is the third major component of the application layer; it is the user interface to the robot end–effector. It provides the user with simple commands (such as "open" and "close") and communicates these commands to the gripper driver (gripper Drv). Note that the gripper interface and the gripper driver may be located on different processors.

jacLib – Jacobian Library. This library contains routines for calculating the Jacobian, Transpose Jacobian, and Inverse Jacobian of the PUMA arm. The equations used by the code are discussed in Appendix B. A complete description of this code is given by [19].

kinLib – Kinematics Library. This library contains routines which calculate the forward and inverse kinematics for the PUMA arm and for the full 9 DOF CIRSSE testbed arm. A detailed description of this code may be found in [12].

pacLib – Position Accommodation Control Library. This is a low level library which implements the force control algorithm described in Chapter 3. The trajectory generator uses the functions in this library to modify position setpoints according to the PAC algorithm. In addition, **pacLib** also contains file–reading functions for the application layer to use.

pumaLib – PUMA Library. This is a low level library which contains basic I/O routines for communicating with the Unimate controller hardware. The PUMA channel driver makes extensive use of these routines.

smLib – This is the State Manager, which was described in the previous section.

spatLib – Spatial Vector Library. This is a math library of functions which operate on vectors containing 6 elements. Such vectors are used in **pacLib**, **jacLib**, etc. to calculate values for each of the 6 degrees of freedom of the robot.

tgen – This is the trajectory generator.

transLib – Transform Library. This math library provides the user with a rich set of functions which operate on homogeneous transforms.

CHAPTER 3
Theory and Implementation of Position Accommodation Force Control

In this chapter, the Position Accommodation Control algorithm is described. The basic theory behind it is given first. Next, several implementation details are provided in order to clarify the concepts presented in the first section. Finally, a number of issues pertaining to this algorithm are discussed.

3.1 Theory

3.1.1 Description

Position Accommodation Control is a force control algorithm in which the forces felt at the end–effector are accommodated by altering the position of the end–effector according to the mass–spring–damper (MSD) equation shown below.

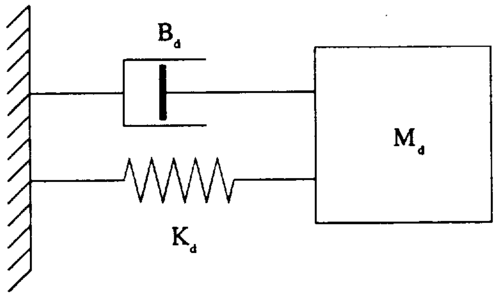


Figure 3.1: Model of Mass–Spring–Damper System

$$M_d\,{}^E\ddot{x}_d + B_d\,{}^E\dot{x}_d + K_d({}^Ex_d - {}^Ex_{ref}) = S(\,{}^Ef_d - {}^Ef\,) \tag{3.1}$$

In the above equation, f_d is the desired force applied to the environment, f is the measured force applied to the environment, and x_{ref} is the reference position for the mass–spring–damper system. All are expressed in the end–effector frame. M_d, B_d, and K_d are the mass, damping and spring terms, respectively; they are

6×6 matrices, usually considered to be diagonal. The remaining terms, with the exception of S, are 6×1 vectors.

S is a 6×6 matrix with a special form. The off–diagonal elements of this matrix are zero, while the diagonal elements are either zero or one. By making a diagonal element equal to one, Position Accommodation is enabled along the corresponding Cartesian axis. If the diagonal element is set to zero, the PAC algorithm will not comply along the corresponding axis, because it will not "see" the force information for that axis. By using this matrix, the user can select the Cartesian directions in which the robot will comply to forces. For this reason, S is commonly called a "selection matrix".

The selection matrix acts like a switch, enabling or disabling compliance along certain Cartesian axes. Otherwise, it does not affect the behaviour of equation (3.1), and therefore, it is fairly transparent to the operation of the PAC algorithm. For this reason, the presence of the selection matrix in the MSD equation will sometimes be ignored in the following discussion.

By making the matrices diagonal, 6 decoupled linear equations are produced, one for each Cartesian degree of freedom of the end–effector. Thus, the motion of the end–effector in each degree of freedom can be made to behave like a *different* mass–spring–damper system. This is a useful ability; in some instances, for example, it may be desirable to comply easily in the tool Z direction, but very little along the X or Y axes.

3.1.2 PAC Is an Integral Force Control Algorithm

When the spring term is set to zero, Position Accommodation Control in an integral force control algorithm [20]. This can be shown fairly easily for the case when the robot is contacting a rigid surface. The general dynamic equation for the robot in contact with a rigid surface is:

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + g(\theta) + J^T f = \tau \tag{3.2}$$

where $M(\theta)$ is the inertia matrix, $C(\theta,\dot{\theta})$ contains the Coriolis and centrifugal terms, and $g(\theta)$ is the load due to gravity. τ is the vector of torques which are applied to the joints via the motors and gear train of the robot. f is the vector of forces which are applied to the environment by the end-effector of the robot.

Consider a position-based joint controller that uses a proportional control law with rate feedback, along with gravity compensation. The equation for this type of controller is:

$$\tau = K_p(\theta_d - \theta) - K_v\dot{\theta} + \hat{g}(\theta) \tag{3.3}$$

Setting equations (3.2) and (3.3) equal to each other results in:

$$M\ddot{\theta} + C\dot{\theta} + g(\theta) + J^T f = K_p(\theta_d - \theta) - K_v\dot{\theta} + \hat{g}(\theta)$$

Because the surface is rigid, it will not comply to a force. Since the robot is in contact with the surface, the joint positions will not change much (i.e. joint motion is negligible). Thus, $\dot{\theta}$ and $\ddot{\theta}$ are approximately zero, and the Jacobian, which is a function of θ, will be approximately constant.

Using this information, and assuming that $\hat{g}(\theta) = g(\theta)$ (i.e. the gravity compensation works well), this equation can be simplified.

$$J^T f = K_p(\theta_d - \theta) \tag{3.4}$$

The term $J^T f_d$ will be added and subtracted from the left side of equation (3.4). In addition, the right side will be expanded. This results in:

$$J^T f - J^T f_d + J^T f_d = K_p\theta_d - K_p\theta \tag{3.5}$$

By making the substitution $f_{err} = f_d - f$, and rearranging terms, the following is obtained:

$$K_p\theta_d = -J^T f_{err} + J^T f_d + K_p\theta \tag{3.6}$$

Because θ is constant, the rightmost two terms in equation (3.6) can be combined, forming a single constant, v. This results in:

$$K_p\theta_d = -J^T f_{err} + v \tag{3.7}$$

At this point, the MSD equation is introduced. Setting $K_d = 0$ in equation (3.1), the following equation is obtained:

$$M_d\ddot{x}_d + B_d\dot{x}_d = f_{err} \tag{3.8}$$

Applying the substitutions $\dot{x} = J\dot{\theta}$ and $\ddot{x} \approx J\ddot{\theta}$ to equation (3.8), produces the following equation:

$$M_dJ\ddot{\theta}_d + B_dJ\dot{\theta}_d = f_{err} \tag{3.9}$$

Multiplying each term in equation (3.9) by $-J^T$, substituting in equation (3.7), results in the following:

$$-J^TM_dJ\ddot{\theta}_d - J^TB_dJ\dot{\theta}_d = K_p\theta_d - v \tag{3.10}$$

Rearranging terms produces the following result:

$$J^TM_dJ\ddot{\theta}_d + J^TB_dJ\dot{\theta}_d + K_p\theta_d = v \tag{3.11}$$

Equation (3.11) is a second order differential equation in θ_d. Provided that the parameters M_d and B_d are chosen properly, θ_d will converge to some constant value. After convergence, $\dot{\theta}_d = \ddot{\theta}_d = 0$, and from equation (3.9), f_{err} also goes to zero. That is, f converges to f_d at the same rate that θ_d converges to a constant.

Note that it is possible that θ_d will have oscillatory behaviour as it cönverges to a constant value. This is possible due to the fact that equation (3.11) is a second order equation. Even if a set of gains is found which allows the desired joint positions to converge without oscillation, the convergence behaviour may change when the robot is put into another configuration. This is because J and J^T are functions of the joint angles, and will change when the robot joints are moved.

By setting M_d to zero, the order of the above equation is reduced:

$$J^T B_d J \dot{\theta}_d + K_p \theta_d = v \tag{3.12}$$

The desired joint angles will converge exponentially, as long as the 6 poles (1 for each desired joint position) are not complex. This is not a difficult constraint to achieve, and it is certainly easier to accomplish than ensuring that the 12 poles resulting from equation (3.11) are not complex. While equation (3.12) is also configuration dependent (due to the Jacobian terms), changing configuration will tend to change the rate of convergence of the desired position (and the force), rather than introducing oscillation.

By setting M_d to zero, we still have an integral force controller. The control law can be derived in a straightforward manner. beginning with the compliance equation:

$$B_d \dot{x}_d = f_{err} \tag{3.13}$$

Substituting $J\dot{\theta}_d$ for $\dot{x}_d$, and rearranging terms, produces:

$$\dot{\theta}_d = J^{-1} B_d^{-1} f_{err} \tag{3.14}$$

By integrating both sides, the following equation results:

$$\theta_d(t) - \theta_d(0) = \int_0^t J^{-1} B_d^{-1} f_{err}(s) ds \tag{3.15}$$

Since the joint positions are constant, $\theta_d(0) = \theta(0) \approx \theta(t)$. In addition, because the Jacobian is constant, it will be moved out from under the integral sign:

$$\theta_d(t) - \theta(t) = J^{-1} B_d^{-1} \int_0^t f_{err}(s) ds \tag{3.16}$$

Substituting equation (3.15) into equation (3.3) produces the final result:

$$\tau = K_p J^{-1} B_d^{-1} \int_0^t f_{err}(s) ds - K_v \dot{\theta} + \hat{g}(\theta) \tag{3.17}$$

Note that equation (3.17) describes the torques applied to the joints by the motors. This equation shows that the PAC algorithm is an integral force controller. In addition to having an integral force term. the control law also contains gravity compensation and damping terms.

3.2 Implementation

3.2.1 Description of Basic Algorithm

This section discusses the components of the robot control system which are involved in the execution of the PAC algorithm. Further discussion of the algorithm is also provided in order to clarify the concepts presented in the previous section.

Figure 3.2 shows the major components of the robot control system (presented in Chapter 2) which are involved in the execution of the Position Accommodation algorithm. The major paths of the data flow are also shown.

During normal operation of these components, regardless of whether the PAC algorithm is being executed, the lowest level support code operates in the following manner. The FTS channel driver reads the force sensor, and places the force measurements into shared memory, available for all tasks to read. The PUMA channel

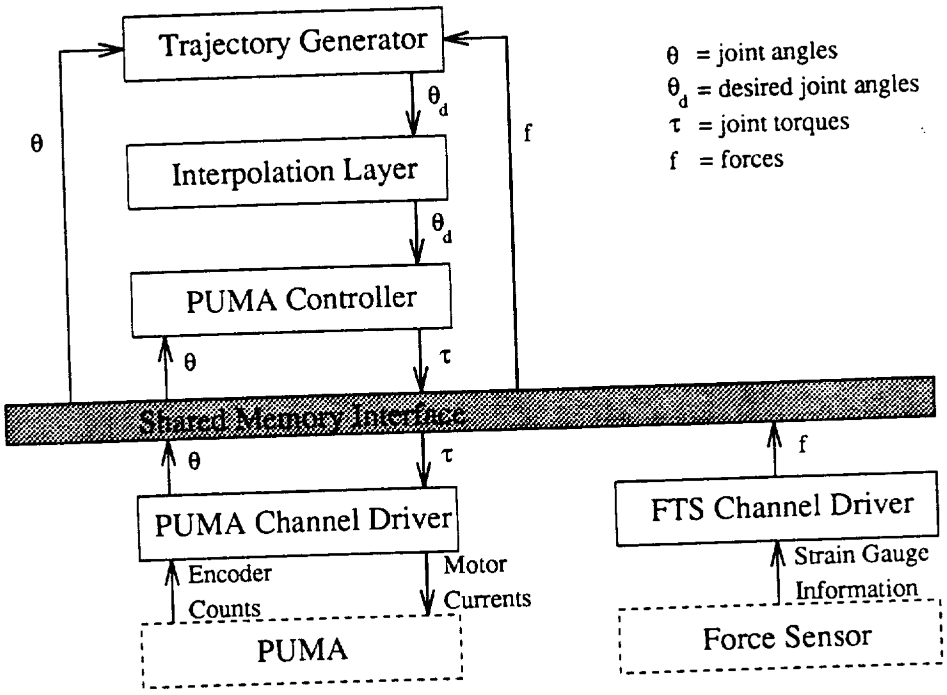


Figure 3.2: Block Diagram of Components Used in PAC Force Control

driver reads the joint positions of the robot and places this data into shared memory as well.

When the system first begins execution of the PAC algorithm, the trajectory generator reads the joint positions and uses the forward kinematics to determine the position of the end-effector in Cartesian space (${}^{0}_{E}T$). This is used as the reference position in all subsequent sampling periods. Recall that Position Accommodation Control requires a reference position (see equation (3.1)) to determine where to move the end-effector.

Every sampling period thereafter, the following operations are performed by the trajectory generator while executing the PAC algorithm:

- The force measurements are read from shared memory.
- The forces are used to calculate a desired setpoint using a mass-spring-damper equation. A homogeneous transform is then generated which describes the

desired end-effector position and orientation with respect to its reference location; that is: ${}^{E}_{E'}T$.

- The desired position and orientation of the end-effector with respect to the world frame of the robot is calculated by using the equation:

$$ {}^{0}_{E'}T = {}^{0}_{E}T \, {}^{E}_{E'}T $$

- New joint angles which will place the end-effector in the desired position are generated using the inverse kinematics for the robot. The desired joint setpoints are then sent to the joint controller (via the interpolation layer).

The joint controller obtains the setpoints from the interpolation layer, and uses its control law to generate motor torques for each of the robot joints. These torques are placed into shared memory for the PUMA channel driver to read and output to the robot hardware.

3.2.2 Second Order Equations

Equation (3.1) expresses the continuous-time equation that describes the PAC algorithm. Since the force controller is implemented on a digital computer, the continuous-time equation must be converted into a discrete-time form. At each time interval, $t = nk, n \geq 0$, the discrete-time equation should behave like its continuous-time counterpart:

$$ M_d \, {}^{E}\ddot{x}_d(nk) + B_d \, {}^{E}\dot{x}_d(nk) + K_d(\, {}^{E}x_d(nk) - {}^{E}x_{ref}) = S(\, {}^{E}f_d - {}^{E}f(nk) \,) \qquad (3.18) $$

In order to simplify the equation, the following substitution will be made for the right-hand side of the equation:

$$ {}^{E}f_{err}[k] = {}^{E}f_d - {}^{E}f[k] \qquad (3.19) $$

To convert the MSD equation into a form which can be easily implemented in the trajectory generator, the following simple-difference approximations for velocity and acceleration will be used:

$$\begin{aligned} \dot{x}_d[k] &\approx \frac{x_d[k] - x_d[k-1]}{T_s} \\ \ddot{x}_d[k] &\approx \frac{\dot{x}_d[k] - \dot{x}_d[k-1]}{T_s} \end{aligned} \tag{3.20}$$

where T_s is the sampling period.

In addition, x_{ref} will be set to zero. This simplifies the equations, but does not limit the control algorithm. The output of the equations presented here is the location of the mass-spring-damper system with respect to its reference position (in this case, zero). This location is then mapped into the location of the end-effector with respect to the end-effector's reference position (${}^{E}_{E'}T$).

Substituting these equations into equation (3.18), the following equation is obtained:

$$\begin{aligned} S\,{}^{E}f_{err}[k] &= M_d \frac{({}^{E}x_d[k] - 2\,{}^{E}x_d[k-1] + {}^{E}x_d[k-2])}{T_s^2} + \\ & B_d \frac{({}^{E}x_d[k] - {}^{E}x_d[k-1])}{T_s} + K_d\,{}^{E}x_d[k] \end{aligned} \tag{3.21}$$

Rearranging terms, the final MSD equation is obtained:

$$\begin{aligned} {}^{E}x_d[k] &= \frac{T_s^2}{M_d + B_d T_s + K_d T_s^2} S\,{}^{E}f_{err}[k] \\ &+ \frac{(2M_d + B_d T_s)}{M_d + B_d T_s + K_d T_s^2}\,{}^{E}x_d[k-1] - \frac{M_d}{M_d + B_d T_s + K_d T_s^2}\,{}^{E}x_d[k-2] \end{aligned} \tag{3.22}$$

Equation (3.22) is the second order equation which is implemented in the Position Accommodation Control library, described in Chapter 2.

3.2.3 First Order Equations

In the case where $M_d = 0$, a first order system results. The continuous-time equation which describes this system is as follows:

$$B_d\,{}^E\dot{x}_d(t) + K_d({}^E x_d(t) - {}^E x_{ref}) = S(\,{}^E f_d - {}^E f(t)\,) \tag{3.23}$$

As before, this equation must be converted into a discrete-time form. The same substitutions will be used for the right-hand side of the equation, and for approximating velocity (equations (3.19) and (3.20)). Substituting these equations into equation (3.23), setting x_{ref} to zero, and converting to discrete-time notation results in:

$$B_d\frac{({}^E x_d[k] - {}^E x_d[k-1])}{T_s} + K_d\,{}^E x_d[k] = S\,{}^E f_{err}[k] \tag{3.24}$$

By rearranging terms, the first order discrete-time equation is obtained. This equation is also implemented in the PAC library.

$${}^E x_d[k] = \frac{T_s}{B_d + K_d T_s} S\,{}^E f_{err}[k] + \frac{B_d}{B_d + K_d T_s}\,{}^E x_d[k-1] \tag{3.25}$$

3.2.4 Translating Forces From the Sensor Frame to The Tool Frame

An important issue which must be handled during the implementation of this algorithm concerns the frame in which the forces are measured and expressed. The forces are measured in the sensor frame, while the robot is moved with respect to the end-effector frame. These frames do not coincide, and may actually be separated by some distance. In order to eliminate this frame problem, it is necessary to "translate" the forces from the sensor frame to the tool frame.

Figure 3.3 depicts two frames connected by a rigid link. Let F denote the force sensor frame, let E denote the end-effector frame, and let the rigid link be the end-effector itself. Suppose there are forces and torques applied to the link

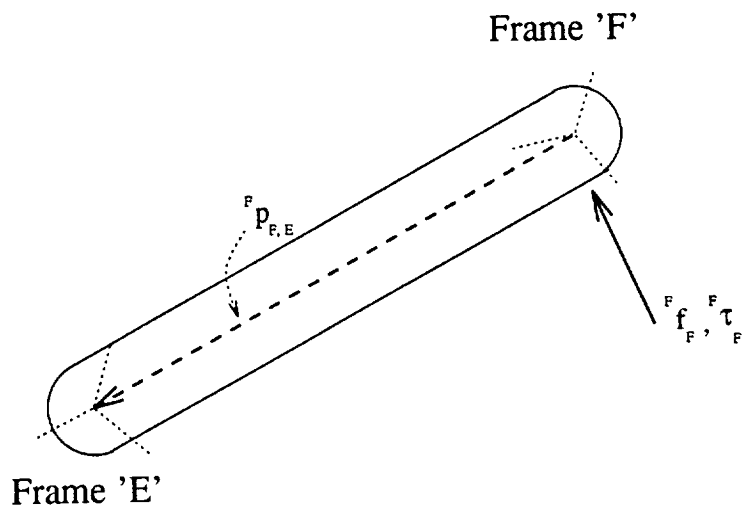


Figure 3.3: Two Frames Connected By a Rigid Link

about frame F, which are measured by the force sensor. These measurements are expressed in the coordinates of frame F. It is desired to find the forces and torques felt at frame E, expressed in frame E.

The forces felt at frame E, but expressed in frame F, can be determined as follows:

$$\begin{aligned} {}^{F}f_{E} &= {}^{F}f_{F} \\ {}^{F}\tau_{E} &= {}^{F}\tau_{F} + {}^{F}f_{F} \times {}^{F}p_{F,E} \end{aligned} \qquad (3.26)$$

Note that f and τ are each 3×1 vectors. The torque equation can be rearranged as follows:

$$\begin{aligned} {}^{F}\tau_{E} &= {}^{F}\tau_{F} - {}^{F}p_{F,E} \times {}^{F}f_{F} \\ &= {}^{F}\tau_{F} + {}^{F}p_{E,F} \times {}^{F}f_{F} \\ &= {}^{F}\tau_{F} + {}^{F}\tilde{p}_{E,F}\, {}^{F}f_{F} \end{aligned} \qquad (3.27)$$

In matrix form, the equations for the force and torque become:

$$\begin{bmatrix} {}^{F}f_{E} \\ {}^{F}\tau_{E} \end{bmatrix} = \begin{bmatrix} I & 0 \\ {}^{F}\tilde{p}_{E,F} & I \end{bmatrix} \begin{bmatrix} {}^{F}f_{F} \\ {}^{F}\tau_{F} \end{bmatrix} \qquad (3.28)$$

To express the forces and torques in frame E, each term is multiplied by the rotation matrix ${}^{E}_{F}R$:

$$\begin{bmatrix} {}^{E}f_E \\ {}^{E}\tau_E \end{bmatrix} = \begin{bmatrix} {}^{E}_{F}R & 0 \\ 0 & {}^{E}_{F}R \end{bmatrix} \begin{bmatrix} {}^{F}f_E \\ {}^{F}\tau_E \end{bmatrix} \tag{3.29}$$

By combining equations (3.28) and (3.29), we have the means for translating the forces and torques in the sensor frame to the end-effector frame:

$$\begin{bmatrix} {}^{E}f_E \\ {}^{E}\tau_E \end{bmatrix} = \begin{bmatrix} {}^{E}_{F}R & 0 \\ 0 & {}^{E}_{F}R \end{bmatrix} \begin{bmatrix} I & 0 \\ {}^{F}\tilde{p}_{E,F} & I \end{bmatrix} \begin{bmatrix} {}^{F}f_F \\ {}^{F}\tau_F \end{bmatrix} \tag{3.30}$$

In order to translate the forces from the force sensor frame to the end-effector frame, it is necessary to have the homogeneous transform describing the position and orientation of one frame in terms of the other. With this information, it is straightforward to modify the forces read by the force sensor before executing the mass-spring-damper equations.

3.2.5 Forward Differencing Versus Backward Differencing

The derivation of the first-order and second-order discrete-time compliance equations involved the use of backward differencing in order to approximate the velocity and acceleration terms (equation (3.20)). Another possible approach to calculating the compliance equations would be to use a forward differencing approximation, such as:

$$\begin{aligned} \dot{x}_d[k] &\approx \frac{x_d[k+1] - x_d[k]}{T_s} \\ \ddot{x}_d[k] &\approx \frac{\dot{x}_d[k+1] - \dot{x}_d[k]}{T_s} \end{aligned} \tag{3.31}$$

Compare these approximations to equation (3.20). In this section, it will be shown that using forward differencing results in a system which is not as robust with respect to parameter variations as the backward differencing result. To illustrate the difficulties with the forward differencing derivation, the first order equation will be rederived. The first order continuous-time equation (ignoring the selection matrix) is:

$$B_d\, {}^E\dot{x}_d(t) + K_d({}^E x_d(t) - {}^E x_{ref}) = {}^E f_d - {}^E f(t) \tag{3.32}$$

Setting x_{ref} to zero, converting to discrete-time notation, and substituting equations (3.19) and (3.31) into equation (3.32) results in:

$$B_d \frac{({}^E x_d[k+1] - {}^E x_d[k])}{T_s} + K_d\, {}^E x_d[k] = {}^E f_{err}[k] \tag{3.33}$$

Rearranging the terms in the equation results in:

$$\frac{B_d}{T_s}\, {}^E x_d[k+1] \;+\; \frac{K_d T_s - B_d}{T_s}\, {}^E x_d[k] = {}^E f_{err}[k] \tag{3.34}$$

Subtracting one from each of the time indices, and rearranging the terms produces:

$${}^E x_d[k] = \frac{T_s}{B_d}\, {}^E f_{err}[k-1] \;+\; \frac{B_d - K_d T_s}{B_d}\, {}^E x_d[k-1] \tag{3.35}$$

Equations (3.35) and (3.25) are clearly different. To determine which is better, the transfer functions for both equations must be determined. Converting equation (3.35) into the Z-domain produces:

$${}^E x_d(z) = \frac{T_s}{B_d} z^{-1}\; {}^E f_{err}(z) + \frac{B_d - K_d T_s}{B_d} z^{-1}\; {}^E x_d(z) \tag{3.36}$$

Rearranging terms results in the transfer function:

$$\frac{{}^E x_d(z)}{{}^E f_{err}(z)} = \frac{T_s z^{-1}}{B_d + (B_d - K_d T_s) z^{-1}} = \frac{T_s}{B_d z + (B_d - K_d T_s)} \tag{3.37}$$

The transfer function for the forward differencing solution has a pole at:

$$z = \frac{K_d T_s - B_d}{B_d}$$

When the pole is inside the unit circle, the simulated spring–damper system will be stable. It is necessary to determine the relative values of the spring and damping terms which will ensure stability. For the system to be stable, $|z| \leq 1$. That is:

$$-1 \leq z \leq 1$$

Substituting in the pole location yields:

$$-1 \leq \frac{K_d T_s - B_d}{B_d} \leq 1$$

Rearranging terms results in the following inequality:

$$0 \leq K_d \leq \frac{2B_d}{T_s} \tag{3.38}$$

The first part of the inequality indicates that K_d must be nonnegative for stability. This holds true for continuous–time systems as well, but for physical springs K is always positive. The second part indicates an additional constraint that K_d can not exceed a threshold which is a function of the damping parameter and the sampling period. This is not paralleled in a physical system; a real mass–spring–damper system is a passive system and is never unstable. At worst, it is marginally stable (when there is no damping).

To provide a comparison, the same analysis will be performed on the first order equation produced using backward differencing (equation (3.25)). This equation is repeated below:

$${}^{E}x_d[k] = \frac{T_s}{B_d + K_d T_s}\,{}^{E}f_{err}[k] + \frac{B_d}{B_d + K_d T_s}\,{}^{E}x_d[k-1] \tag{3.39}$$

Converting to the Z-domain results in:

$$^{E}x_d(z) = \frac{T_s}{B_d + K_d T_s}\, {}^{E}f_{err}(z) + \frac{B_d}{B_d + K_d T_s} z^{-1}\, {}^{E}x_d(z) \tag{3.40}$$

Rearranging terms results in the transfer function:

$$\frac{^{E}x_d(z)}{^{E}f_{err}(z)} = \frac{T_s}{(B_d + K_d T_s) - B_d z^{-1}} = \frac{T_s z}{(B_d + K_d T_s) z - B_d} \tag{3.41}$$

The transfer function for the backward differencing solution has a zero at the origin, and a pole at:

$$z = \frac{B_d}{B_d + K_d T_s}$$

As before, in order for this system to be stable, $|z| \leq 1$. For this system, there are no positive values of B_d and K_d which will cause it to go unstable. It is straightforward to show this, starting again with the criteria for stability.

$$-1 \leq z \leq 1$$

Substituting in the pole location produces:

$$-1 \leq \frac{B_d}{B_d + K_d T_s} \leq 1$$

Rearranging terms results in the following inequality:

$$\frac{-2B_d}{T_s} - K_d \leq 0 \leq K_d \tag{3.42}$$

Thus, $K_d \geq 0$ for stability. The first part of the of the inequality can be rewritten as:

$$\frac{2B_d}{T_s} + K_d \geq 0$$

which can be rearranged to produce:

$$B_d \geq \frac{-K_d T_s}{2} \tag{3.43}$$

Since K_d and T_s are nonnegative, the second requirement indicates that the damping parameter must be chosen greater than some negative value (which is a function of the spring term and sampling period). Because the damping term is normally chosen positive, this is not a restriction. Thus, if $B_d \geq 0$, $K_d \geq 0$, the system will be stable. This result agrees well with a physical mass–spring–damper system.

From this comparison of the two transfer functions, it is evident that backward differencing is a better strategy to use. The system equation that is produced by backward differencing more closely resembles a physical, continuous–time system, and is more robust with respect to varying the parameters.

3.3 Discussion

There are a number of other issues which must be considered when implementing the Position Accommodation algorithm. Several of these issues are discussed in this section.

3.3.1 Force Sensor

The PAC equations which describe the motion of the end–effector use the symbol f as the measured force which is applied to the environment. It is important to realize that the force sensor actually measures forces applied to the sensor itself, *not* to the environment. It is necessary, therefore, to negate the force measurement from the sensor at some point in the coded algorithm.

It should also be noted that in any type of force control, the robot will only react to forces which it can perceive. Forces applied to the links of the arm below the location of the force sensor will not be noticed by the PAC algorithm (although the joint controllers will act to reject the disturbance force and maintain the joint positions). Forces applied to the force sensor or gripper will be picked up by the force sensor and acted upon by the trajectory generator according to the mass–spring–damper equations.

Lastly, it should be kept in mind that the force sensor is a physical device, and will have noise. Thus, the force measurements obtained by the sensor will fluctuate slightly about the actual force value. Consider what would happen if the parameters K_d and f_d are set to zero in the PAC algorithm. From Section 3.1.2, we know that in such a situation, the PAC algorithm is an integral force controller. If the robot is not contacting a surface, in theory the forces measured by the sensor are zero, and the end–effector will maintain its position indefinitely. In practice however, the end–effector will begin to drift slowly in the directions which have no spring value. This is because of the small fluctuations in the force measurements; these non-zero force readings have been mapped into end–effector motions by the PAC algorithm.

3.3.2 Singularities

Robots typically have a number of *singular* points (or *singularities*), whose location is dependent upon the geometry of the robot arm. These singular points are places in the robot joint space where the robot loses a degree of freedom. At a singular point, the robot loses the ability to move in a certain direction. Consider, for example, joint 5 of the PUMA. When this joint is at 0 degrees, the axes of joints 4 and 6 coincide, causing the robot to lose a degree of freedom. Because of the alignment of these joints, the gripper can not rotate about the Cartesian axis perpendicular to both joints 4 and 5.

The PAC algorithm has difficulty with robot singularities. The problem lies in its use of inverse kinematics to map the desired Cartesian position of the end–effector to the desired positions of the robot joints. If the robot is too close to a singular position, small Cartesian motion will result in large movement of some of the robot joints.

Equation (3.17) provides another insight into the problem that the PAC algorithm has with singularities. Note that the integral force term is scaled by the inverse Jacobian. At singularities, the Jacobian loses rank, while the inverse Jacobian blows up. Near the singular points of the robot, the torques applied to the joints will become very large, resulting in large joint motion.

In order to handle the problem of motion near singularities while executing the PAC algorithm, several options are available. These include:

- *Avoid singularities* – This requires the identification of the singularities for the robot being used, or some path planning algorithm which will inherently avoid singularities. While the singularities for the PUMA are well known [19, 21, 22], it may be difficult to determine the singular points for less common robot types.

- *Limit the maximum joint velocities* – this will prevent the joints from moving faster than some acceptable limit, at the expense of causing the gripper to deviate from the desired trajectory corresponding to the MSD motion.

3.3.3 PAC Architecture

Because the PAC algorithm is implemented in the trajectory generator, it is isolated somewhat from the dynamics of the robot itself. The joint controller is left with the responsibility of handling the low level motion of the robot. This structure has important ramifications. One benefit is that different joint controllers can be used to control the joints. For example, instead of using a PD controller (as was done for this research), a PI, PID, or sliding mode controller might also be used. A related result is that the PAC force control algorithm can be implemented relatively easily on industrial robots. Such robotic systems often have position-based joint controllers, and allow users to program trajectory generators to provide the controller with setpoints.

Another point which should be kept in mind is that the force control algorithm is run with the sampling period of the trajectory generator. Typically, this might be in the range of 20–40 milliseconds, and is much slower than the controller servo rate. This sampling rate will limit the range of mass, spring and damper parameters which can be simulated by the robot system, and therefore will limit the response time of the system.

In the previous discussion of the PAC algorithm, f_d was considered a constant. However, it might also be implemented as a function of time. This would result in a controller which could track a force "trajectory", instead of just servoing to a constant force value.

In the discussion of the implementation, the reference point of the PAC algorithm was considered to be the position of the end–effector when the algorithm was started. As an alternative, a "moving" reference point might be used. The trajectory generator's regular function is to take knotpoints from the application layer, and generate setpoints every sampling period. If these setpoints are used as the reference points, the robot can follow a desired trajectory until the gripper contacts an object or surface. The gripper would then comply to the forces according to the MSD equations.

3.3.4 Effect of Spring Term In PAC Algorithm

In Section 3.1.2, it was shown that when K_d was eliminated, the PAC algorithm became an integral force controller. Setting $K_d = 0$ was necessary for this to occur. This is because the spring force serves to oppose the force error, which is the input to the mass–spring–damper equation. The spring parameter serves to limit the distance that the mass will move away from its reference position.

If the spring parameter is non–zero, the actual force in steady state will not equal the desired force. To see this, consider the scenario where there is a spring force. (The selection matrix will be ignored for this discussion.) The equation governing the overall motion of the end-effector is:

$$M_d \, {}^E\ddot{x}_d + B_d \, {}^E\dot{x}_d(t) + K_d({}^E x_d(t) - {}^E x_{ref}) = {}^E f_d - {}^E f(t) \tag{3.44}$$

Assuming the parameters are chosen to allow the robot to stably contact the environment, the system will achieve a steady state force value. At this point, $\ddot{x}_d = \dot{x}_d = 0$, and the following equation results:

$${}^E f(t) = {}^E f_d - K_d({}^E x_d(t) - {}^E x_{ref}) \tag{3.45}$$

Thus, the steady state force is not equal to the desired force; there is a difference which is proportional to the distance that the gripper has moved away from its starting point.

As the spring constant for a given axis is increased, the distance that the gripper will deflect for a given force will decrease. If K_d is increased to infinity, the gripper will not alter its position at all in response to a force. This is equivalent to not allowing compliance in that degree of freedom.

3.4 Summary

In this section, the theory and implementation of Position Accommodation Control has been discussed. The basic concept of the algorithm was presented first, and it was shown that the algorithm is an integral force controller. A description of the basic tasks which must be accomplished during execution of the PAC algorithm was presented, and the discrete-time equations which are implemented in the algorithm were provided. Finally, a number of important issues concerning the PAC algorithm were discussed.

CHAPTER 4
Theory and Implementation of Direct Force Servoing

In this chapter, the Direct Force Servoing algorithm is described. The basic theory behind it is presented first. Next, some implementation details are provided in order to clarify the concepts presented in the initial section. Finally, several issues pertaining to this force control algorithm are discussed.

4.1 Theory

4.1.1 Description

Direct Force Servoing is a force control algorithm in which the forces exerted by the end–effector on the environment are controlled by directly controlling the torques applied to the joints of the robot. In contrast to the Position Accommodation algorithm, no position setpoints are generated. In order to relate the end–effector forces to the joint torques, the transpose of the Jacobian is used. It has been shown that the transpose of the Jacobian, J^T, relates the forces felt at the end–effector to the torques felt at the joints of the robot [13].

The control equation for the DFS algorithm is:

$$\tau = J^T F + \hat{g}(\theta) \tag{4.1}$$

where $\hat{g}(\theta)$ is the gravity compensation torque. The control signal, F, is defined as:

$$F = K_P(f_d - f(t)) + K_I \int_0^t (f_d - f(s))ds + K_{f_d} f_d \tag{4.2}$$

In the above equation, f_d is the desired force to be applied to the environment, and f is the measured force applied to the environment. Both are 6×1 vectors. The terms K_P, K_I, and K_{f_d} are 6×6 diagonal matrices which contain the control gains. By making the matrices diagonal, 6 decoupled control equations are produced.

It is also useful to incorporate a selection matrix, S, in this control algorithm. It functions in the same manner as the selection matrix in the PAC algorithm. Equation (4.2) then becomes:

$$F = S\left(K_P(f_d - f(t)) + K_I \int_0^t (f_d - f(s))ds + K_{f_d} f_d \right) \tag{4.3}$$

The selection matrix allows the algorithm to control forces only along certain Cartesian directions, ignoring the directions which are not enabled. Because the selection matrix is transparent to the behaviour of the controller in the directions which are enabled, much of the following discussion will ignore the presence of the matrix.

4.1.2 DFS Does Not Produce Straight–Line Cartesian Motion

Unlike Position Accommodation Control, the DFS algorithm presented here does not move the gripper in straight lines in Cartesian space. This is a limitation of the algorithm presented here; however, for small motions (less than a centimeter) this control method provides a reasonable approximation to straight–line Cartesian motion.

To show this feature of the Direct Force Servoing algorithm, we start with the general dynamic equation for a robot in contact with a rigid environment (equation (3.2)), repeated below:

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + g(\theta) + J^T f = \tau \tag{4.4}$$

Assuming the motions of the robot are slow, the Coriolis term is negligible, and can be ignored. By substituting the DFS control law, equation (4.1), into equation (4.4), and canceling the gravity terms, the following equation is obtained:

$$M\ddot{\theta} + J^T f = J^T F \tag{4.5}$$

Rearranging the terms, and applying the approximation $\ddot{x} \approx J\ddot{\theta}$, produces:

$$MJ^{-1}\ddot{x} = J^T(F - f) \tag{4.6}$$

The final result is obtained by rearranging terms:

$$\ddot{x} = JM^{-1}J^T(F - f) \tag{4.7}$$

The matrix formed by $JM^{-1}J^T$ is typically nondiagonal; therefore, force terms along one Cartesian axis will produce accelerations in other Cartesian directions. To see this more clearly, consider the following scenario.

Let us assume that the only forces which are desired or are actually being applied are in the tool Z direction. Thus, $F = \begin{bmatrix} 0 & 0 & F_z & 0 & 0 & 0 \end{bmatrix}^T$, and $f = \begin{bmatrix} 0 & 0 & f_z & 0 & 0 & 0 \end{bmatrix}^T$. (This could be achieved using the selection matrix described earlier.) If the controller produces straight–line Cartesian motion, the only end–effector motion would be in the tool Z direction. However, J, J^T, and M^{-1} are all nondiagonal matrices. Thus, there will be accelerations in Cartesian directions other than the tool Z direction. Therefore, the DFS algorithm does not move the gripper in straight–line paths in Cartesian space.

4.1.3 Direct Proportional Force Control Is Not Robust With Respect To Time Delay

Another characteristic of the Direct Force Control algorithm which has been noted in [7] is that direct proportional force control tends to be unstable with respect to time delay. To show this result, consider again the general dynamic equation for a robot:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + g(\theta) + J^T f = \tau \tag{4.8}$$

Assuming that the joint motions are slow, we make the approximation that $\ddot{\theta} = \dot{\theta} = 0$. Substituting the DFS control law, equation (4.1), into equation (4.8), and canceling the gravity terms, we obtain:

$$J^T f = J^T F \tag{4.9}$$

Thus, we have that $f = F$. That is, the force on the environment is equal to (in theory) the force control signal, F.

Now consider the discrete–time control law:

$$\tau[k] = J^T F[k-1] \tag{4.10}$$

where the F signal is generated using a purely proportional force equation:

$$F[k] = K_P(f_d - f[k]) \tag{4.11}$$

Note the single delay in equation (4.10). This is representative of the fact that control calculations take a finite amount of time. Because of the needed calculation time, discrete–time controllers commonly calculate the control torque during one sampling period, and output it during the *next* sampling period.

Combining equation (4.10) with the discrete–time version of the relation between end–effector forces and joint torques: $\tau[k] = J^T f[k]$, the following relationship is obtained:

$$f[k] = F[k-1] \tag{4.12}$$

Substituting equation (4.11) in the above relationship results in:

$$f[k] = K_P(f_d - f[k-1]) \tag{4.13}$$

Rearranging terms produces the following equation:

$$f[k] + K_P f[k-1] = K_P f_d \tag{4.14}$$

Converting the equation into the z-domain results in the following:

$$f(z)(I + K_P z^{-1}) = K_P f_d(z) \tag{4.15}$$

Recall that K_P is a diagonal matrix, so the above matrix equation contains 6 decoupled equations. The transfer function of any of the component systems is:

$$\frac{f_i(z)}{f_{d_i}(z)} = \frac{K_{P_i}}{1 + K_{P_i} z^{-1}} = \frac{K_{P_i} z}{z + K_{P_i}}, \quad 1 \leq i \leq 6 \tag{4.16}$$

Note that in order for each system to be stable, $|K_{P_i}| < 1.0$, for $1 \leq i \leq 6$. While this limit assumes that the surface is rigid (recall that we started with an equation which assumed the robot was in contact with a rigid surface), it nevertheless points out that direct proportional force control may not be robust due to the time delay inherent in a physical system.

4.2 Implementation

4.2.1 Description Of Basic Algorithm

This section discusses the components of the robot control system which are involved in the execution of the DFS algorithm. Further discussion of the algorithm is provided in order to clarify the concepts presented in the previous section.

Figure 4.1 shows the major components of the robot control system which are involved during the execution of the Direct Force Servoing algorithm.

The lowest level support code is in continual operation, regardless of whether the DFS algorithm is running. The FTS channel driver obtains the forces from the force sensor and places the measurements into shared memory. Similarly, the PUMA channel driver reads the joint positions of the robot and stores the values in shared memory.

During execution of the Direct Force algorithm, the controller must perform the following operations every sampling period.

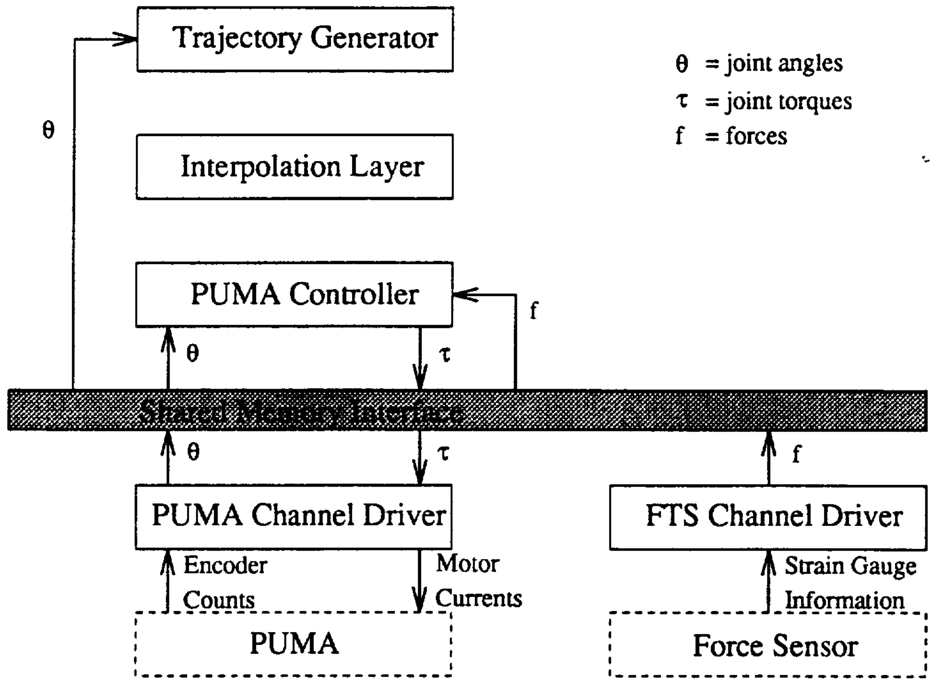


Figure 4.1: Block Diagram of DFS Force Control

- The force and position measurements are read from shared memory.
- The gravity compensation torques are calculated based on the position of the robot joints.
- The force control signal (F) is calculated based upon equation (4.2).
- The joint torques are calculated from F using the transpose Jacobian.
- The gravity compensation torques are added into the joint torques and the values are stored in shared memory.

The PUMA channel driver will read the joint torques from shared memory during the next time sample, and output them to the robot hardware.

It should be noted that the trajectory generator does not actually participate in the control loop. However, when the system first begins execution of the DFS algorithm, the trajectory generator must be notified. This is because the trajectory generator normally provides the controller with position setpoints. The joint controller does not need these setpoints while it is executing the DFS algorithm.

It is especially important to notify the trajectory generator when the system is to leave the Direct Force control mode. Upon leaving this mode, the joint controller will again require position setpoints. The trajectory generator must determine the new position of the robot upon leaving the force control mode, so that the setpoints it gives to the joint controller do not cause a sudden change in the position of the robot.

4.2.2 Discrete–Time Equations

Equations (4.1) and (4.2) comprise the continuous-time version of the DFS control law. The discrete-time equations (with the selection matrix) are shown below.

$$\tau[k] = J^T F[k-1] + \hat{g}(\theta) \tag{4.17}$$

where the control signal, F, is defined as:

$$F[k] = S\,(\;K_P(f_d - f[k]) + K_I \sum_{i=0}^{k} (f_d - f[i]) + K_{f_d} f_d\;) \tag{4.18}$$

These equations are implemented in the Direct Force Servoing library, described in Chapter 2.

4.3 Discussion

There are a number of issues which should be considered when implementing the Direct Force Servoing algorithm. Several of these issues are discussed in this section.

4.3.1 Force Sensor

In Chapter 3, several points about the force sensor were mentioned, in the context of Position Accommodation Control. These issues pertain to the Direct Force Servoing algorithm as well. For example, as with the PAC algorithm, it is necessary to translate the forces from the force sensor frame to the end–effector frame for direct force control. The other topics (negating the force sensor measurements, what forces the sensor can perceive, and sensor noise) are equally applicable to the DFS algorithm.

4.3.2 Singularities

In contrast to the behaviour of the PAC algorithm, the DFS algorithm should not have difficulty with robot singularities. This is because the control law uses the transpose Jacobian to relate the joint torques to the forces at the end–effector. The transpose Jacobian does not exhibit the same problems as the inverse kinematics when the PUMA arm goes into a singular position. At a singular point, the arm loses a degree of freedom, and may not be able to achieve the desired force in some direction. However, no large motion of the joints should occur.

4.3.3 DFS Architecture

The DFS algorithm is implemented in the joint controller for the robot. Thus, unlike the PAC algorithm, it is not isolated from the dynamics of the robot. The location of the algorithm prevents easy implementation of this algorithm in industrial robotic systems, because the joint controller itself must be specialized enough to handle forces. However, because of its location, the DFS algorithm runs at the controller servo rate. In a typical robotic system, the controller may run about an order of magnitude faster than the trajectory generator. Because of this faster sampling rate, it is reasonable to expect that a Direct Force controller would be more responsive than a Position Accommodation controller.

It should also be noted that in the earlier discussion of the Direct Force algorithm, f_d was considered to be a constant. As an alternative, f_d could be implemented as a function of time. This would result in a controller which could track a force "trajectory".

4.3.4 Gravity Compensation

The term "gravity compensation" refers to a process by which the gravitational forces applied to the robot links are canceled. This is accomplished by reading the robot joint positions every sampling period, calculating the torques which are being applied to the joints by gravity, and then applying an opposing motor torque to counteract the effect of gravity. If the gravity compensation is done properly, the arm will remain motionless when the brakes holding the joints are released.

It is interesting to note that if the control gains K_P, K_I, and K_{f_d} in equation (4.2) are all zero, then F is zero as well. In this case, the DFS algorithm reduces to pure gravity compensation control. Thus, gravity compensation control can be thought of as a "special case" of the Direct Force Servoing algorithm presented here.

4.4 Summary

In this chapter, the theory and implementation of Direct Force Servoing has been discussed. The basic algorithm was presented first, and it was shown that the DFS controller is not a Cartesian controller. In addition, it was shown that direct proportional force control should not be robust with respect to time delay. Next, a discussion of the basic tasks which must be accomplished by the controller during execution of the DFS algorithm was presented, and the discrete-time equations which were implemented were provided. Lastly, a number of issues pertaining to the DFS algorithm were discussed.

CHAPTER 5
Results of Force Control Experiments With Flexible Surfaces

5.1 Motivation

In order to perform useful work, a robot must come into contact with its environment. In order to contact the the environment safely, some means of controlling the forces exerted upon the environment must be used. This is the motivation for force control; to allow the robot to safely come in contact with the environment.

The robot environment may contain a variety of different materials. Because each material will have its own flexibility, the robot force control algorithm must be able to deal with surfaces of different compliance in order to operate safely. Therefore, it is necessary to develop an understanding of how environmental flexibility affects the stability of the force control algorithm.

This chapter describes a series of experiments performed in order to gain an understanding of the effects that surface flexibility has upon force control algorithms. It is divided into the following sections: Section 5.2 discusses the materials which were used to test the force control algorithms, and section 5.3 describes the experiments performed. Sections 5.4 and 5.5 discuss the results of the PAC and DFS experiments, respectively. Finally, a brief summary is provided.

5.2 Model of Environment

A simple model of the environment used by many researchers ([7, 9], etc.) is that of a mass–spring–damper. This model has been used to accurately predict results in force control experiments [9], and therefore has some experimental justification.

The experiments that were performed focused on two of the parameters which

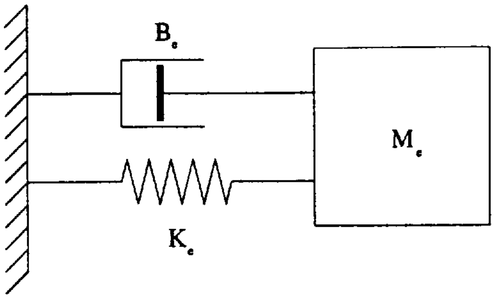


Figure 5.1: Model of Environment

		stiffness (K)	
		low	*high*
damping (B)	*low*	plastic ball	aluminum
	high	plastic lid	wood

Table 5.1: Relative flexibility of surfaces used in contact experiments

characterize the motion of the environment: stiffness and damping. In order to investigate how these parameters affect the stability of the force control algorithm, four surfaces were chosen for testing. They are shown in Table 5.1. The selected materials are fairly common and their flexibility varies considerably.

Aluminum has high stiffness and low damping. The material is not easy to deform, but may vibrate when struck. Wood has high damping and high stiffness. Like aluminum, it is difficult to deform, yet will not vibrate readily when hit (motions are damped out). A plastic can lid exhibits low stiffness, since it is relatively easy to deform the lid. The lid is also highly damped; when the force is removed, the lid takes time to return to its original position. Finally, the plastic ball has low stiffness and low damping because it deforms easily, but returns to its original shape quickly when released.

5.3 Description of Experimental Setup

Figures 2.3 and 2.4 in Chapter 2 depict the test setup used for all force control experiments. All compliant surfaces that were tested were supported on a steel table. This table is extremely rigid and was firmly supported on steel rails. Likewise, the 3 DOF platform which supports the PUMA is also extremely rigid. Thus, the most flexible part of the environment was the compliant surface that was being tested.

The basic force control experiment went as follows. Using a position-based joint level PID controller, the robot arm was positioned so that the gripper was about 1.5 millimeters above the surface to be tested. A set of force control parameters was specified; the force control algorithm was then started, and was directed to apply a force of 10 Newtons to the surface being tested. In order to simplify the experiments, only forces in the tool Z direction (pointing outward from the tool tip) were used in the force control calculations.

This basic experiment was repeated several hundred times. Each of the four surfaces described above was tested using both force control algorithms. For each control algorithm, one or more parameters was varied to determine its range of stability, that is, what values the parameters could take on and still have the gripper stably apply the desired force to the surface.

5.3.1 Varying Force Control Parameters

For the Position Accommodation Control algorithm, in order to have the gripper apply the desired force to the environment, the spring term was set to zero. The initial PAC experiments consisted of two distinct sets of trials, in which one parameter was held constant, while the other was varied. The two parts went as follows:

- The damping term was set to a conservative (high) value, and the mass term was increased from zero.

- The mass term was set to zero. The damping term was then set fairly high (for slow motion), and lowered until the robot could no longer contact the environment in a stable manner.

For the Direct Force Servoing algorithm, two parameters were also varied. Similar to the PAC experiments, the DFS experiments were divided into two parts, in which one control gain was set to zero, while the other gain was varied.

- K_P was set to zero, and K_I was increased until the robot–environment system went unstable (or the forces generated exceeded a safety threshold).

- K_I was set to zero, and K_P was increased until the system went unstable.

In both parts of the DFS experiments, K_{f_d} was set to zero.

5.3.2 Followup Experiments

In order to eliminate the effect of impact force on the control algorithm (recognized to be a problem by [4, 6, 9]), another set of experiments was performed. The robot was positioned as before, and was commanded to apply a 1 Newton force to the surface being tested, using gains which were found to be stable in the initial experiments. After the actual force had settled to the desired value, the desired force and the control gains were then switched to an experimental set which directed the robot to apply a 10 Newton force to the environment. Essentially, the robot was switched from one force control routine (with one set of parameters) to a second routine with a different set of parameters. This eliminated the effect of the impact spike on the performance of the control algorithm, because the gripper was already in contact with the surface when the experimental gains were being tested.

The above gain–switching technique was performed with both force control algorithms, and all surfaces were tested. In these experiments, the mass term was set to zero for the PAC algorithm. Only the damping term was varied. For the

DFS algorithm, these experiments were divided into two parts exactly as the initial experiments were. In each part, one control gain (K_P or K_I) was varied while the other was set to zero.

5.3.3 Use Of Gripper

For all force control experiments, the gripper held a cylinder of aluminum 1.25 inches in diameter and 1 inch long. There were several reasons for this. Due to the geometry of the robot fingers, a cylinder of this size caused the fingers to angle back away from the cylinder. Thus, when the gripper moved down to contact a surface, the cylinder is the part that touched the surface, instead of the gripper fingers. The smaller area of the cylinder more closely approximated a point contact.

In practice, it is likely that the gripper would be holding an object while trying to exert a force. An example of this is a robot using a piece of chalk to write on a blackboard. The robot applies a force to the blackboard through the chalk. Because the gripper will often be used in the actual application, it was logical to include the use of the gripper in the force control experiments as well.

5.3.4 Sampling Rates

In order to be able to make some comparison between the PAC and DFS algorithms, the following sampling rates were chosen for the main components of the research code. The FTS channel driver was run with a period of 9.0 ms, the joint controller was run at 4.5 ms, and the trajectory generator was run at 9.0 ms. These sampling rates were held constant for all experiments, and for both control algorithms. In DFS mode, the controller read the forces twice as often as the trajectory generator did in PAC mode. However, the forces were only updated every 9 ms, so both algorithms obtained new force information at the same rate. Because the joint-level controller was run at a sampling rate of 4.5 milliseconds for both force control modes, the torques applied to the joints by the motors (in both modes) were updated every 4.5 ms.

5.4 Experimental Results – Position Accommodation Control

5.4.1 Varying The Mass Term

Figures 5.2 – 5.5 illustrate some typical results obtained during the first PAC experiments. In the plots shown, the PAC algorithm is being used to contact aluminum; the damping has been set to 200 Newtons/$\frac{\text{meter}}{\text{second}}$, and the mass term was increased from 0.0 kg to 1000.0 kg.

Several things can be observed from these plots. In Figure 5.2, the steady state force error is zero. That is, the actual force converged to the desired value of 10 Newtons. This indicates that the force error is being integrated, and supports the idea presented in Chapter 3 that the PAC algorithm is an integral force control method.

Another important feature in Figure 5.3 is the difference in the desired and actual Cartesian positions of the end effector. The desired position is in fact *inside* the object that the gripper is touching; the robot gripper cannot move to the desired position because the material it is contacting is in the way. This difference in Cartesian space implies that there is a difference between the desired and actual values of the joint angles. The PD controllers (recall that joint PD control is used during Position Accommodation Control) are applying torques to the joints in an effort to drive the joint positions to their setpoints. Because the environment is in the way of the gripper, these joint torques cause the gripper to apply a force to the environment. If the desired position leaves the surface (moves out of the object), the gripper will also tend to leave the surface. Figures 5.4 and 5.5 illustrate this.

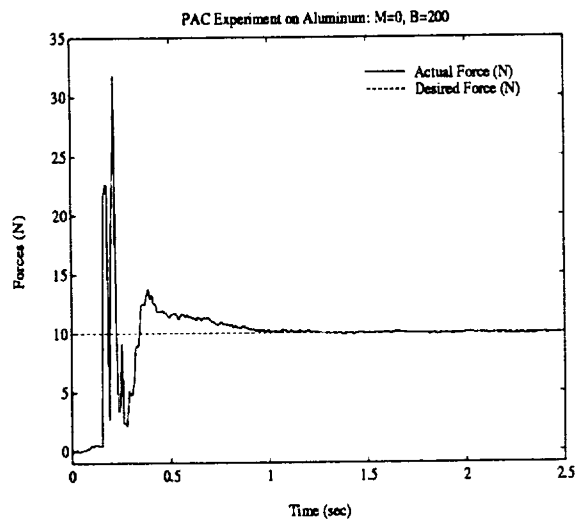
Note that when the desired position leaves the surface of the environment, the actual position of the gripper never lifts off of the surface to the degree specified by the desired position setpoint (Figure 5.5). This can be explained by friction; it is known that significant joint friction is present in the robot which was used to test the force algorithms. Since the joint controllers are PD-type, it is not unreasonable to expect some position tracking problems. In spite of this, the PAC algorithm is able to achieve zero steady-state force error (when the parameters M_d and B_d are appropriately selected).

The joint friction provides natural damping for the system, and in some ways may even be beneficial. For example, in Figure 5.7 ($B = 100.0$) at time $t = 0.5$ seconds, the desired position leaves the surface of the material being contacted temporarily. Due to friction however, the gripper does not leave the surface.

Figures 5.2 - 5.5 illustrate an interesting trend; as the mass increases, oscillations begin, with the robot gripper "bouncing" slowly on the surface. This phenomenon occurred for all materials tested. Note that as the mass is increased, the force (and position) oscillations increase in amplitude and decrease in frequency.

This result is not entirely unexpected. It was pointed out in Chapter 3 that the mass term may cause oscillations in the forces. Although the assumptions made in Section 3.1.2 are only approximated in reality, the point of the argument is still valid: adding a mass term tends to complicate the behaviour of the PAC algorithm, without improving performance.

The oscillation behaviour that was observed can be explained in an intuitive manner: before the gripper contacts the environment, the force error causes the mass-damper system simulated in the trajectory generator to move toward the surface. The actual position of the gripper follows the desired position until the gripper contacts the environment; at which time, force is built up. However, the large mass being simulated limits the acceleration (how fast the velocity can change) of the desired position setpoint. Thus, the desired position maintains its direction of motion even though the actual position may not (due to the surface blocking the



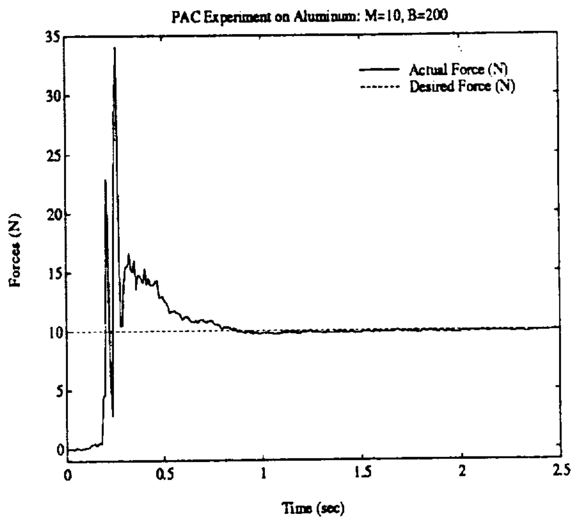
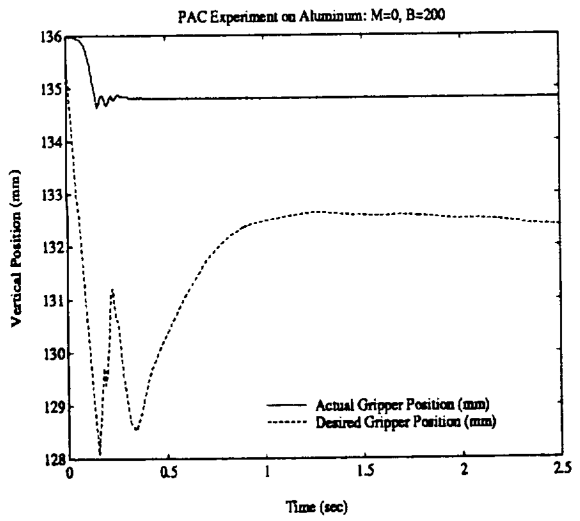


Figure 5.2: Effect of Increasing Mass Term in Position Accommodation Control: Force Profiles



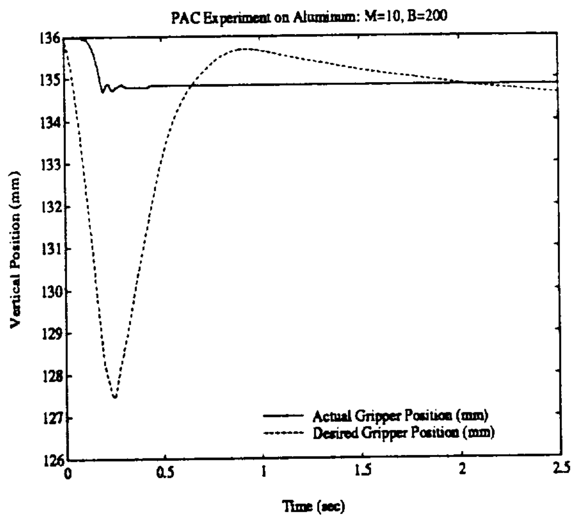
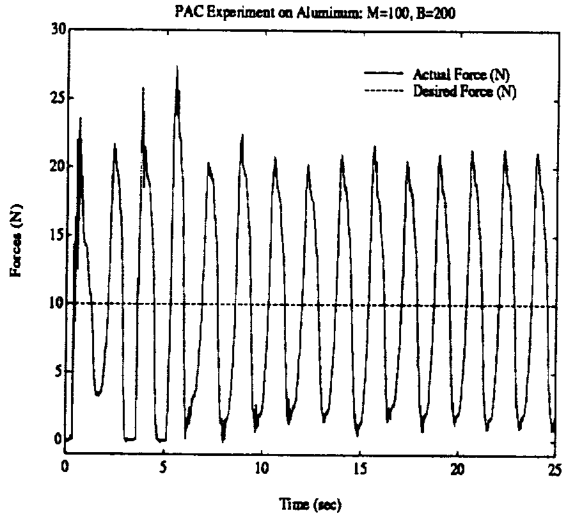


Figure 5.3: Effect of Increasing Mass Term in Position Accommodation Control: Position Profiles



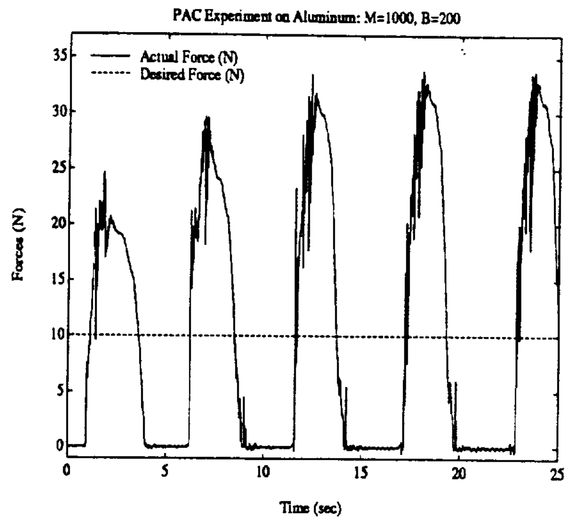
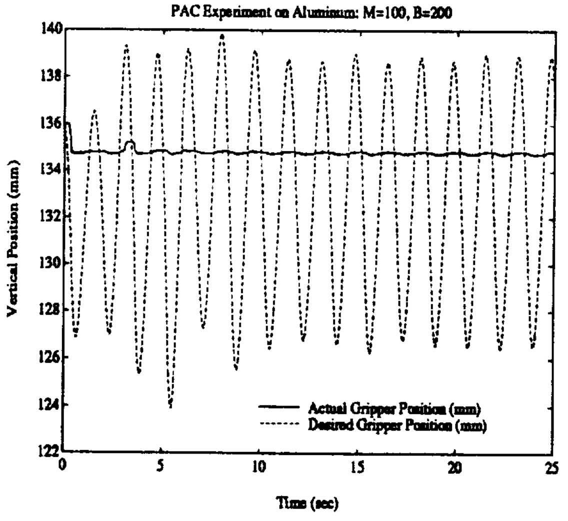


Figure 5.4: Effect of Increasing Mass Term in Position Accommodation Control: Force Profiles



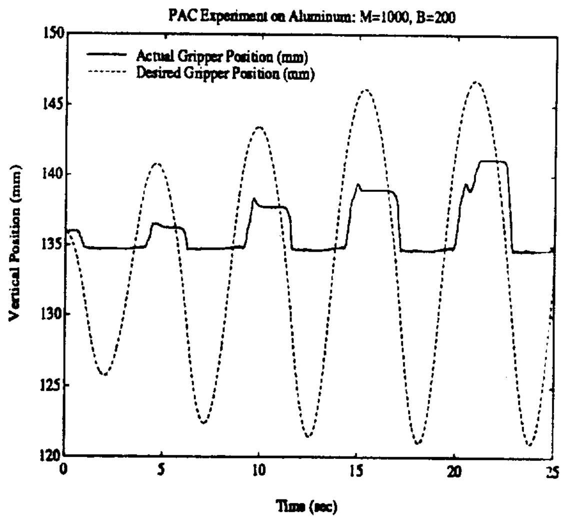


Figure 5.5: Effect of Increasing Mass Term in Position Accommodation Control: Position Profiles

gripper's motion). As the position error increases, so too does the force exerted on the environment.

After enough time passes, the force error builds up sufficiently to change the direction of the simulated mass, and the desired position returns to the surface of the environment. As the desired position approaches the actual position, the force on the surface decreases. Again however, the large simulated mass becomes a problem. The desired position is moving away from the surface, and the force error cannot change the direction of motion quickly enough to prevent this. When the desired position leaves the interior of the environment surface, the actual position will follow, causing the gripper to lift off the surface.

Once the gripper is off the surface, the force measurements become zero, and after some time, the force error will build up enough to direct the simulated mass toward the surface again. The cycle then repeats; the result is a series of slow bounces on the surface. Further increases to the mass term will reduce the acceleration of the desired position setpoint still further, resulting in slower oscillations (because the larger mass cannot change direction as quickly) but with higher peaks in the force readings.

In order to eliminate oscillations due to the simulated mass term, it is preferable to use low mass or even zero mass in the PAC algorithm. This has been suggested by [7, 20]; the results presented here have confirmed the theoretical results presented by these researchers.

5.4.2 Varying The Damping Term

In the second set of PAC experiments, the mass term was set to zero, while the damping term was varied. Tables 5.2 and 5.3 show the lowest stable damping terms determined during these experiments. Damping terms larger than the ones shown were stable, while values smaller than those shown were unstable, causing uncontrolled bouncing and/or excessive forces.

surface	Damping (B)
plastic lid	67.25
plastic ball	44.5
aluminum	68
wood	70

Table 5.2: Lowest stable PAC damping values found for various surfaces. Gripper started approximately 1.5 mm above surface.

surface	Damping (B)
plastic lid	67.25
plastic ball	44.0
aluminum	58
wood	60

Table 5.3: Lowest stable PAC damping values found for various surfaces. Gripper started on surface exerting a force of 1 Newton.

The tables indicate that the range of PAC parameters may vary widely depending upon the surface being contacted. Thus, the type of surface that the robot contacts *does* affect the stability of the Position Accommodation algorithm. Therefore, this force control algorithm must take into account the type of surface being contacted.

Figures 5.6 – 5.9 illustrate what happens when the damping term is lowered. The figures clearly show that as the damping term is decreased, the initial force spike (which occurs when the gripper contacts the environment) increases. Intuitively, this trend makes sense.
The Cartesian velocity of the gripper is determined by the force error divided by the damping. As the damping decreases, the velocity of the gripper will increase. When the gripper impacts the surface at higher velocities, the resulting force spike is higher

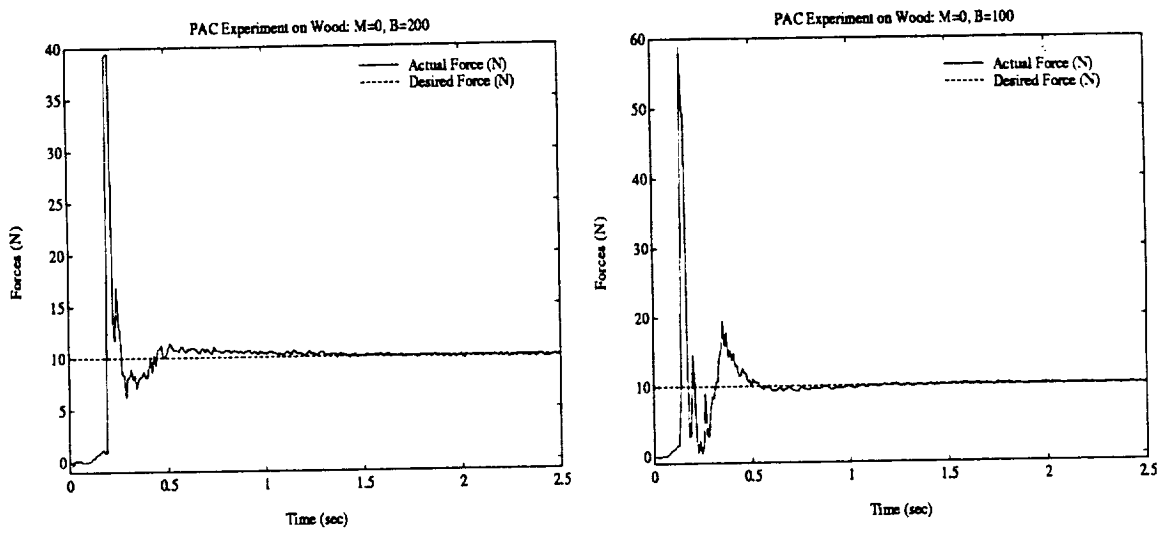


Figure 5.6: Effect of Lowering Damping Term in Position Accommodation Control: Force Profiles

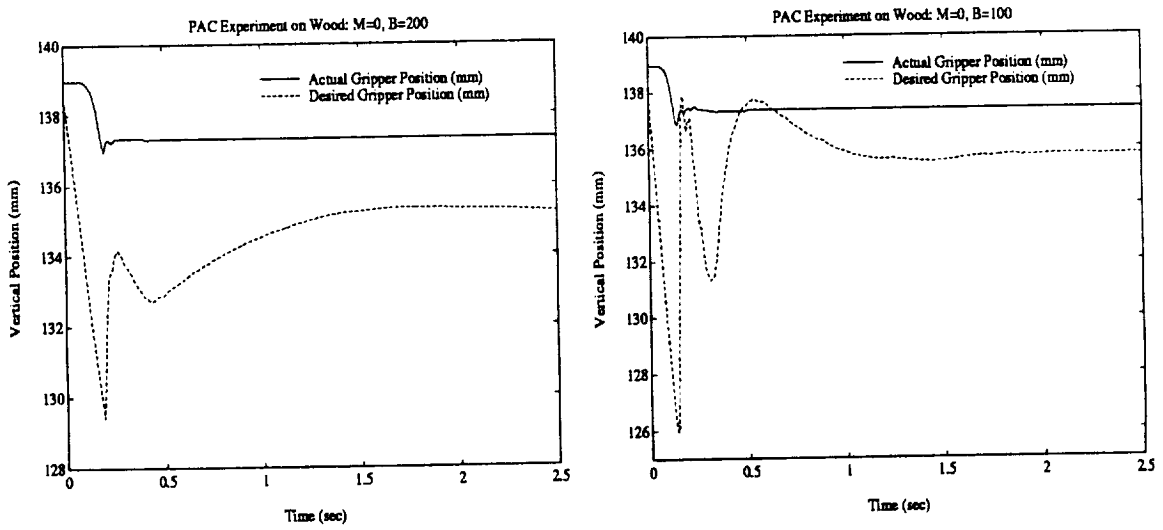


Figure 5.7: Effect of Lowering Damping Term in Position Accommodation Control: Position Profiles

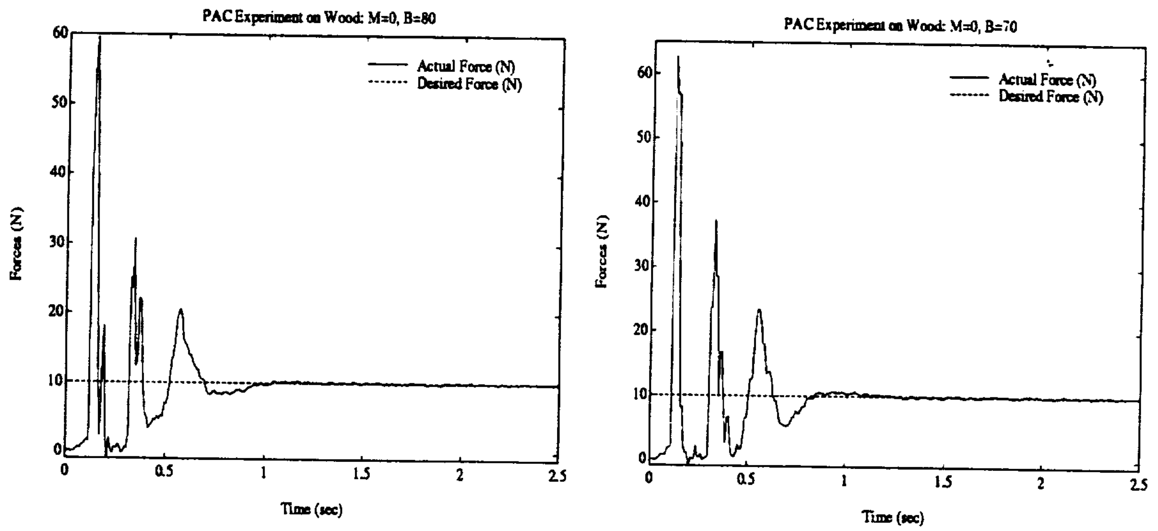


Figure 5.8: Effect of Lowering Damping Term in Position Accommodation Control: Force Profiles

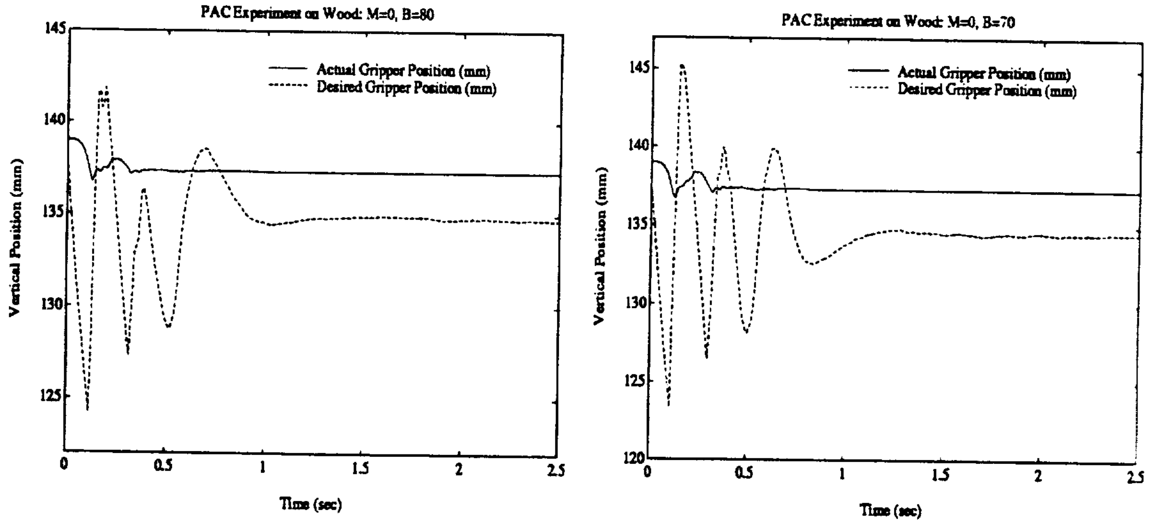


Figure 5.9: Effect of Lowering Damping Term in Position Accommodation Control: Position Profiles

as well. As one would expect, this pattern occurred for all surfaces, indicating that the phenomenon is not dependent upon the surface being impacted.

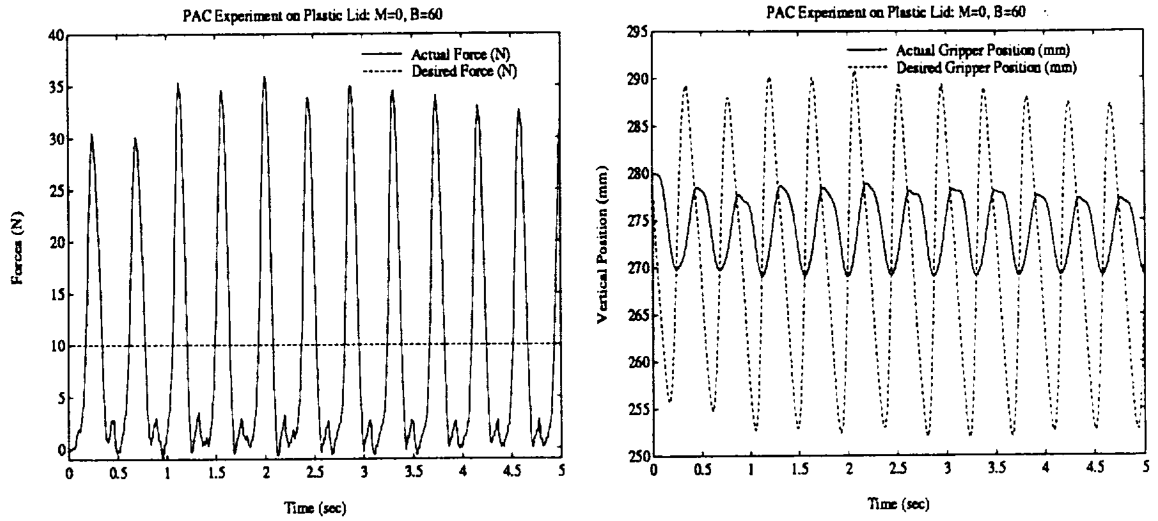


Figure 5.10: Lowering Damping Term Too Far Will Cause Instability in PAC Algorithm

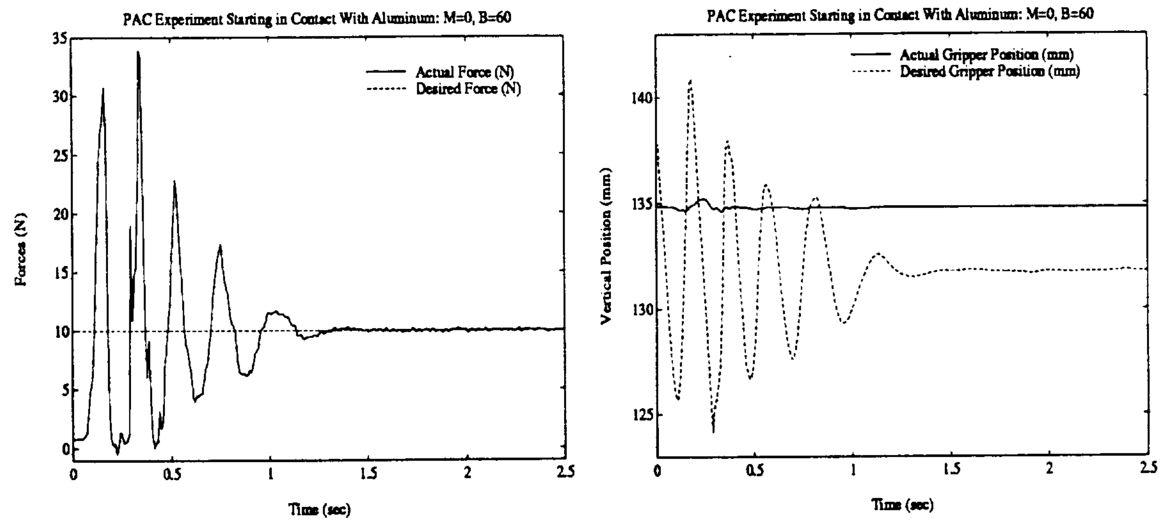


Figure 5.11: PAC Damping Term May Be Reduced By Starting In Contact With Surface

Figure 5.10 shows the effect of reducing the damping term beyond the stability region for the surface being contacted. In this case, the gripper was started above the plastic lid. High frequency oscillations set in, causing large force spikes.

For stiff surfaces, the damping term could be reduced further when the PAC algorithm was started while in contact with the surface. Figure 5.11 illustrates this for aluminum. Table 5.3 shows the reduced damping gains. Note that the lowest stable damping gain (see Table 5.2) when the gripper was started above the aluminum surface was 68.0.

When the gripper was started in contact with the more flexible surfaces, the damping terms could not be reduced much further. Notice that when the gripper started in contact with the plastic lid, the damping term could not be reduced at all.

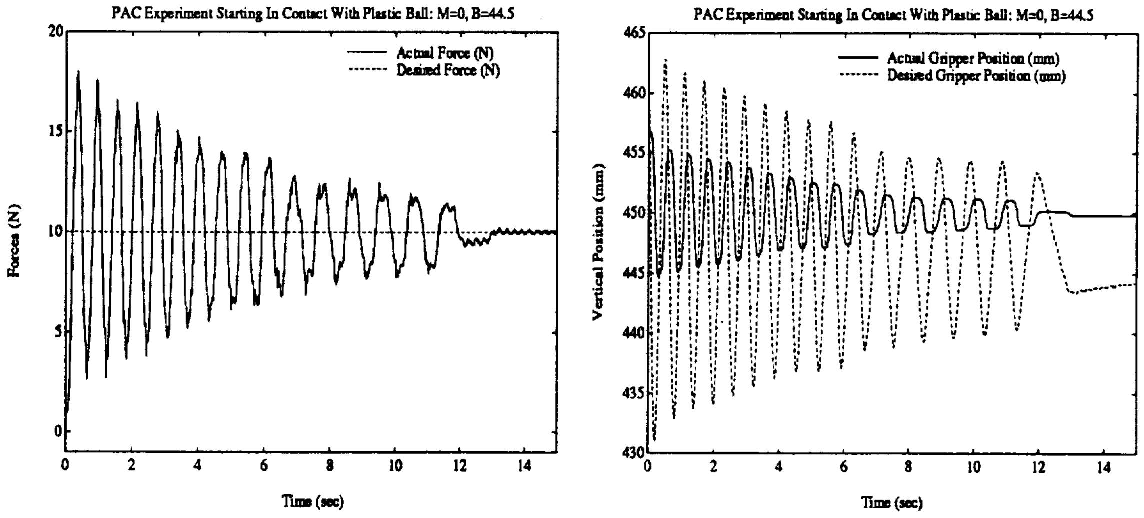


Figure 5.12: Damping Term Can Be Reduced Much More When Contacting Plastic Ball

Finally, it should be noted when the robot contacted the ball, the damping term could be reduced well below that for any of the other surfaces. Figure 5.12 shows the performance of the PAC algorithm on the plastic ball, with a damping gain of 44.5. This is much smaller than the minimum value for any of the other surfaces. Although the response is very oscillatory, it is stable.

5.5 Experimental Results – Direct Force Servoing

Several interesting results were obtained by the series of DFS experiments described above. They will be presented in this section, along with a number of illustrative plots.

Tables 5.4 and 5.5 summarize the range of stable gains determined during the experiments. Gains larger than those shown caused instability (either bouncing with increasing amplitude of oscillation, or excessive forces which caused safety code to stop the experiment).

surface	K_P $(K_I = 0)$	K_I $(K_P = 0)$
plastic lid	0.0 - 4.8	0.0 - 16.2
plastic ball	0.0 - 5.7	0.0 - 10.6
aluminum	0.0 - 1.52	0.0 - 30.0
wood	0.0 - 1.52	0.0 - 33.0

Table 5.4: Range of DFS gains for which force control is stable. Gains were varied separately. Gripper started 1.5 mm above surface.

surface	K_P $(K_I = 0)$	K_I $(K_P = 0)$
plastic lid	0.0 - 7.0	0.0 - 16.6
plastic ball	0.0 - 6.3	0.0 - 12.7
aluminum	0.0 - 2.20	0.0 - 70.0
wood	0.0 - 1.84	0.0 - 66.0

Table 5.5: Range of DFS gains for which force control is stable. Gains were varied separately. Gripper started on surface exerting a force of 1 Newton.

The tables indicate that the range of stable gains varies considerably for the different surfaces. This held true for both the proportional and integral direct force control experiments. Thus, the type of surface that the robot contacts *does* affect the stability of this force control algorithm. Therefore, the DFS algorithm must also take into account the type of surface being contacted.

5.5.1 Direct Integral Force Control

The integral force control algorithm performed similarly on both the aluminum and wood surfaces. This is reflected in the fact that the range of stable gains for both materials are so similar. Figures 5.13 and 5.14 illustrate the performance of the direct force servoing algorithm on wood. For these plots only integral control was used, and the gripper was started above the surface of the wood. Note that as the integral gain is increased, the response time of the system improves; when $K_I = 5.0$, the robot requires above 1.5 seconds in order to achieve the desired force. When $K_I = 20.0$, the desired force is achieved in about 0.5 seconds.

Another phenomenon which can be seen in these plots is the initial force spike which occurred upon contact of the gripper with the environment. As K_I was increased, the force spike also increased. This phenomenon has also been noted by other researchers [9]. Note that when $K_I = 30.0$, the gripper lifted off the surface after the initial force spike.

This bouncing is due to the integral action. When the algorithm first starts, the gripper is above the surface. Thus, the force on the environment is zero, and the force error is non–zero. This error is integrated and scaled to produce the force control signal, which is then converted into joint torques. Note that before contact, the control signal (and hence the joint torques) is the highest for the $K_I = 30.0$ case. Thus, the joints are being driven harder than in all of the previous cases, which results in the greatest impact force. Because of the integral control and high gain, the control signal changes rapidly in response to the force spike and causes the gripper to pull off the surface slightly before settling.

As the gain is increased, eventually a point is reached where the force spike is large enough to cause the gripper to pull away from the surface to the same height that it started from. This is the point of marginal stability; in this situation, a series of force spikes would occur, each causing the gripper to pull away from the surface to the starting height. Increasing the gain beyond this point would cause the system to become unstable because each bounce would be higher than the last, causing ever-increasing force spikes. The instability phenomenon is illustrated in Figures 5.15, when the gripper was contacting the ball.

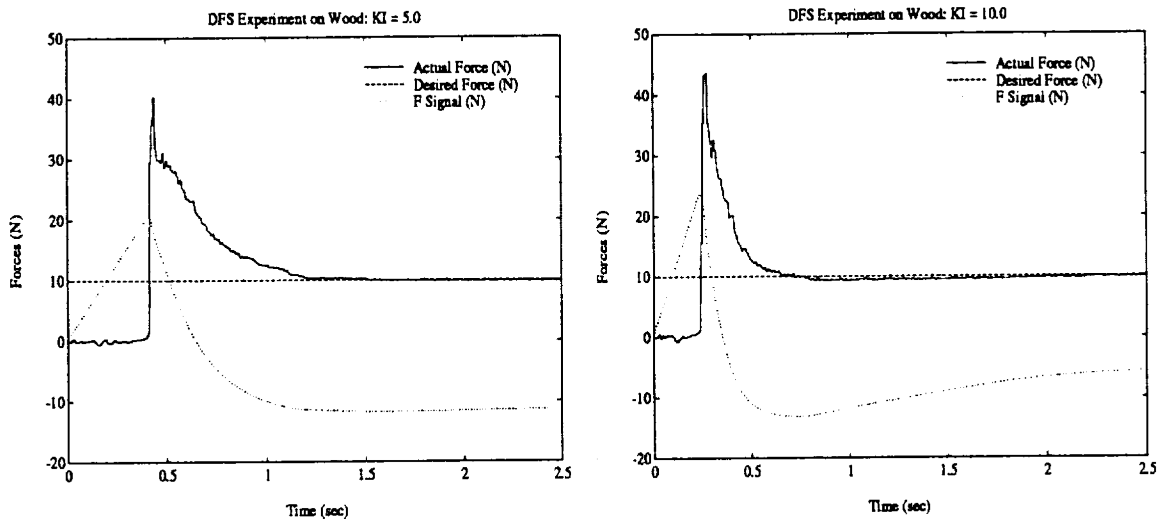


Figure 5.13: Performance Of Direct Integral Force Control Algorithm on Wood: $K_I = 5.0, 10.0$

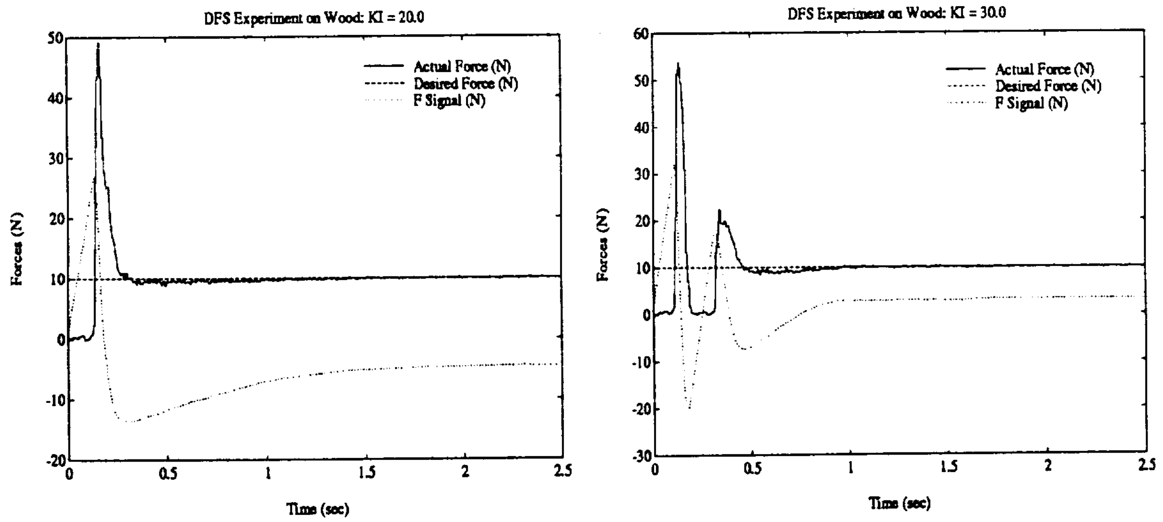


Figure 5.14: Performance Of Direct Integral Force Control Algorithm on Wood: $K_I = 20.0, 30.0$

A comparison of Tables 5.4 and 5.5 indicates that impact force is a problem.

For every material, the integral gains could be increased when the gripper was started in contact with the surface. For the stiffer surfaces, the integral gain could be increased dramatically. An example of this is shown in Figure 5.16. Note that for this run, the integral gain is set at twice the maximum value that was found to allow the gripper to contact the aluminum block. The plot shows that when the algorithm was started, the force on the aluminum block was 1 Newton. The system was stable, and achieved the desired force of 10 Newtons in less than half a second.

While raising the integral gain higher will provide faster response time, there is a potential hazard in doing so. With very high integral gains the robot-environment system is stable only as long as the gripper stays in contact with the environment. If the gripper leaves the surface (due to a disturbance, for example), excessive integral gains will cause the control algorithm to drive the gripper back to the surface, causing large impact forces.

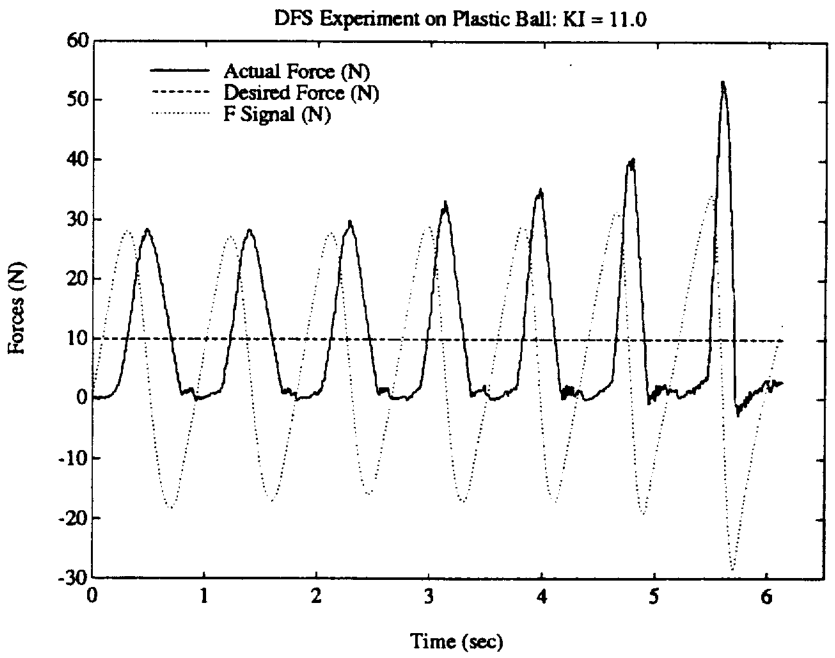


Figure 5.15: Integral Force Gain Is Severely Limited on Plastic Ball

The tables also indicate that the range of stability for the integral gains is

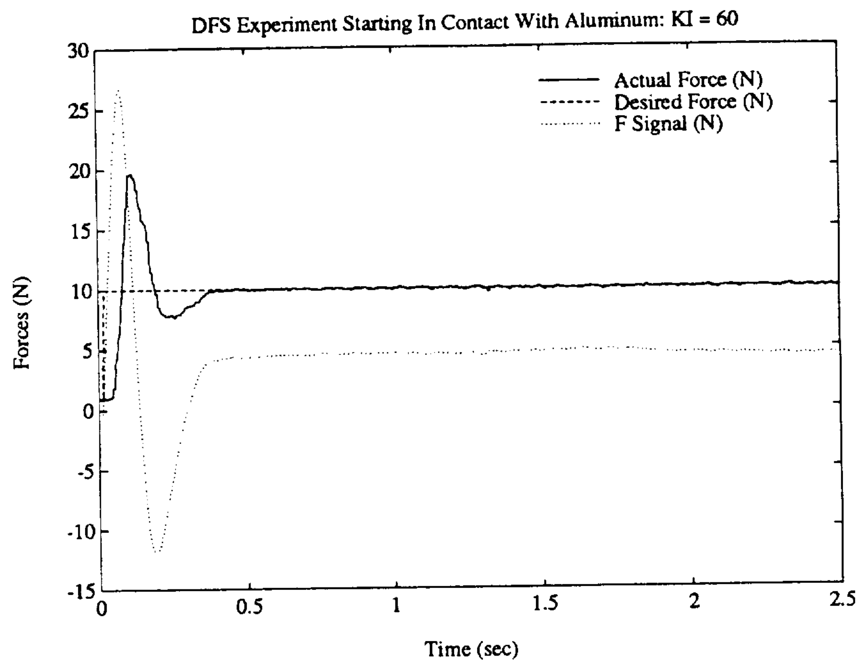


Figure 5.16: Integral Gain Can Be Increased When Starting In Contact With The Aluminum Surface

much wider on stiffer surfaces than on more compliant surfaces. Figures 5.17 and 5.18 show the effect of an integral gain of 15.0 when the lid was contacted. Note the oscillations in the force and position graphs. Even though the gain was low, the system exhibited damped oscillations. Comparing this with Figure 5.14 (K_I = 20.0), it is apparent that the higher integral gain is not as appropriate to use when the environment is more flexible.

The performance of direct integral force control when contacting the ball was even worse than when contacting the plastic lid. Figures 5.19 and 5.20 show the performance of the system when integral force control was used on the ball, with a gain of 10.0. Again, it is apparent that the integral control does not perform as well on the softer surface. When a gain of 11.0 was used to contact the ball, the system went unstable (Figure 5.15).

Figure 5.19 also shows another problem. After 5 seconds, the actual force is still not equal to the desired force. Note that the control signal is winding up after

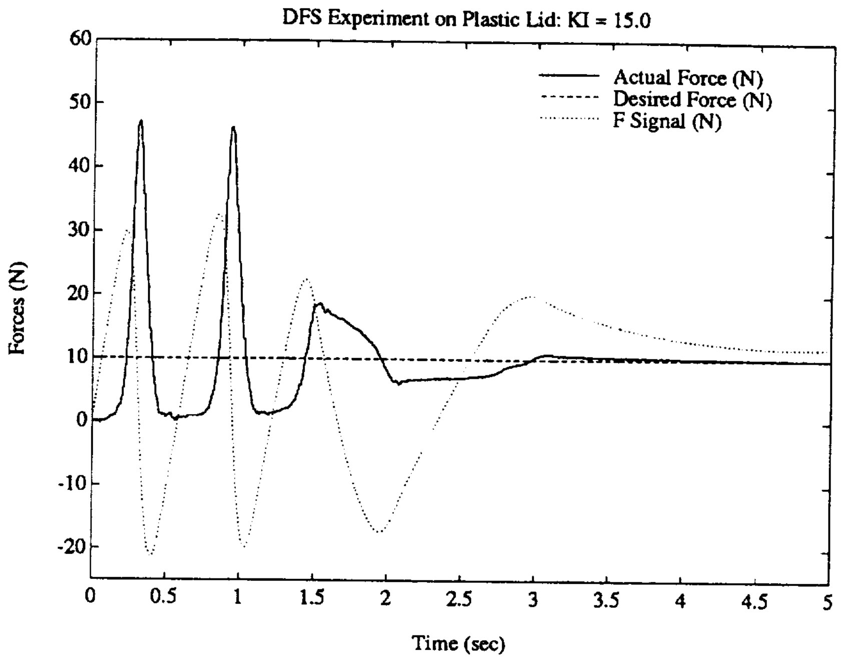


Figure 5.17: Performance Of Basic Integral Force Control Algorithm on Plastic Lid: Force Profile

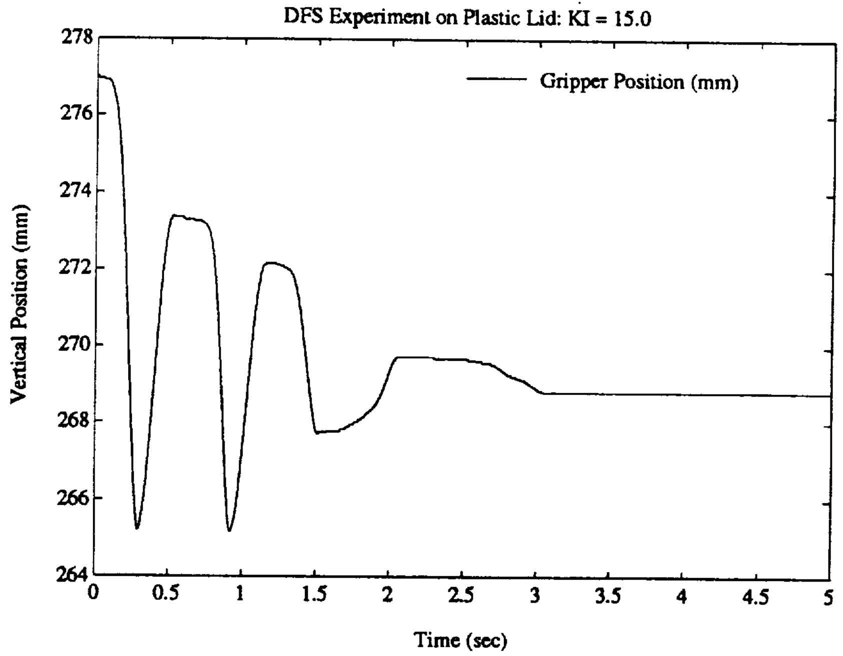


Figure 5.18: Performance Of Basic Integral Force Control Algorithm on Plastic Lid: Position Profile

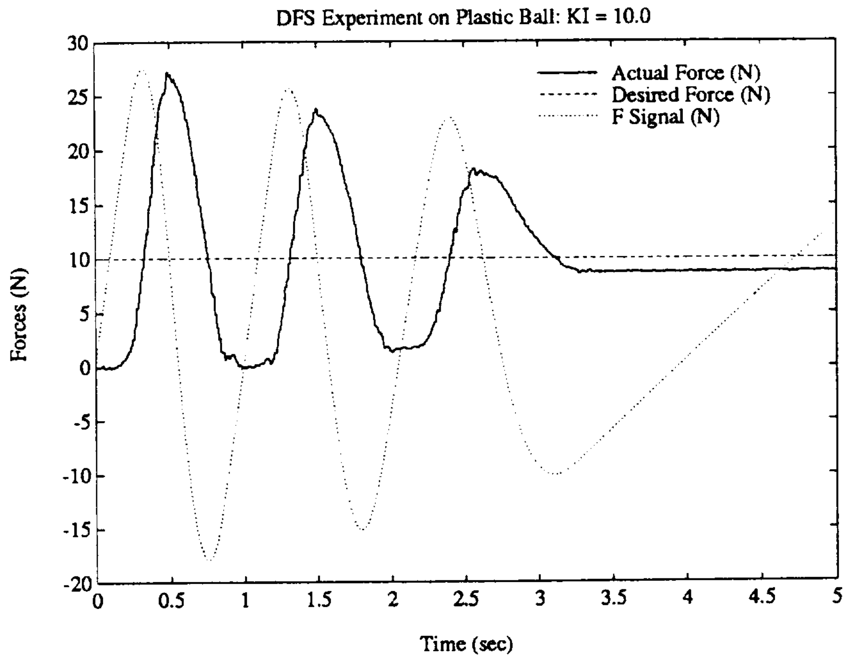


Figure 5.19: Performance Of Direct Integral Force Algorithm on Plastic Ball: Force Profile

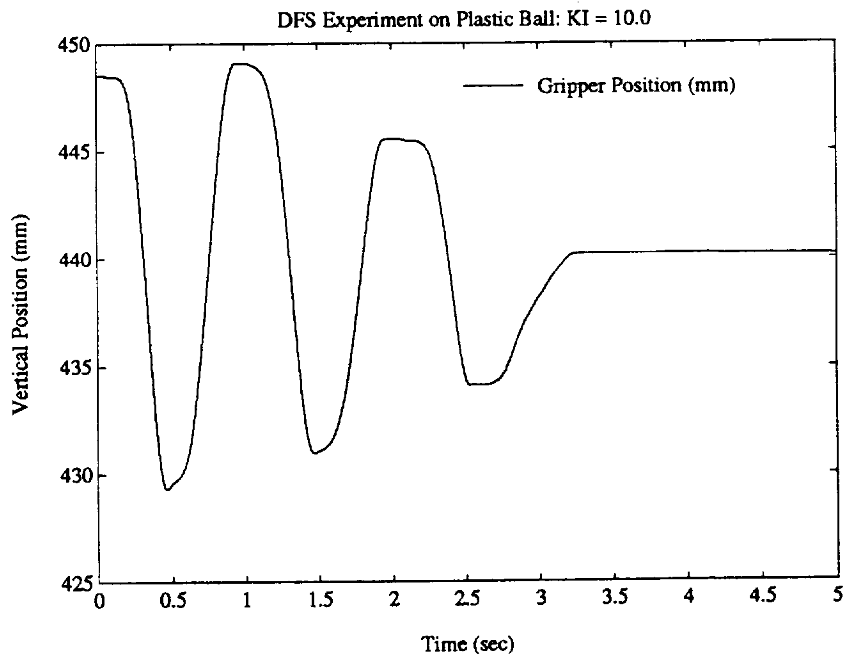


Figure 5.20: Performance Of Direct Integral Force Algorithm on Plastic Ball: Position Profile

time $t = 3$ seconds. From the position plot, it can be seen that even though the control signal (and hence, the joint torques) is changing, the position of the gripper is not moving. This phenomenon is due to the joint friction mentioned previously. In this case, the joint friction has prevented the arm from moving, and the torque has to wind up in order to overcome it. For this test run, the desired force was achieved by time $t = 6.5$ seconds.

5.5.2 Direct Proportional Force Control

As was discussed in Chapter 4, pure proportional force control is not as robust as integral force control. Evidence for this is given in Tables 5.4 and 5.5; the range of stable proportional gains is much more limited than the range of stable integral gains. This was found to be true for all surfaces. Figure 5.21 shows the result of increasing the proportional gain beyond the limit set for aluminum. High frequency bouncing was observed, with very large force spikes. Notice that the proportional gain for this case is only 1.7.

In addition to the more limited stability range, pure proportional control also exhibits steady state error. This can be seen from Figure 5.22; after the transient disappears, the force value hovers around 7.5 Newtons. This is entirely expected from classical control theory; this feature, along with the more limited range of stable gains makes proportional control less attractive than integral control.

As with integral control, the proportional gains could be increased when the control algorithm was started in contact with the environment. Thus, impact forces also contribute to the force control problem when direct proportional force control is used. Figure 5.22 illustrates an example of this phenomenon. Notice that the proportional gain is 2.0, which is higher than the maximum stable gain when the gripper started above the surface ($K_P = 1.52$ for aluminum).

It should be noted from the tables that the range of stability for the direct

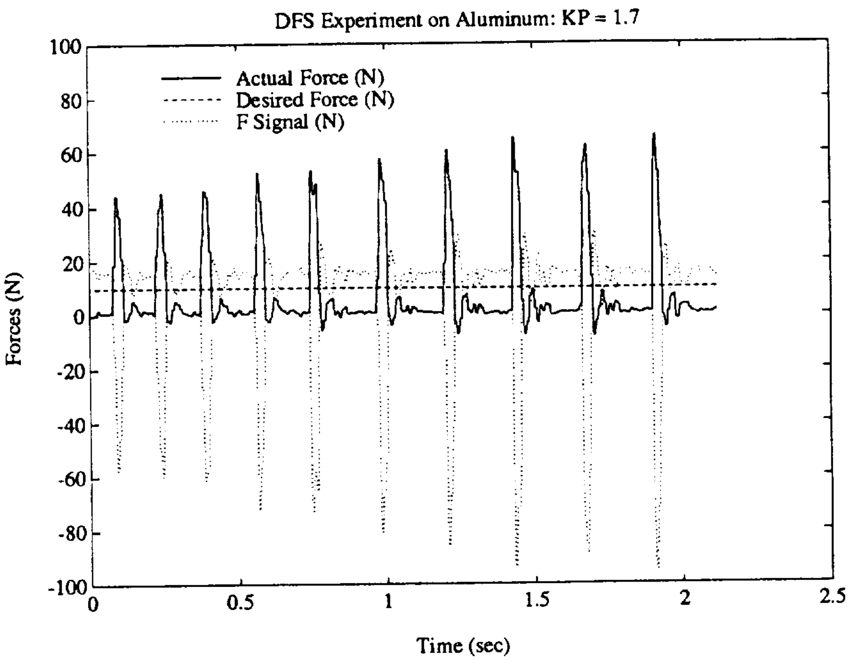


Figure 5.21: Small Proportional Gains May Cause Instability On Aluminum: Force Profile

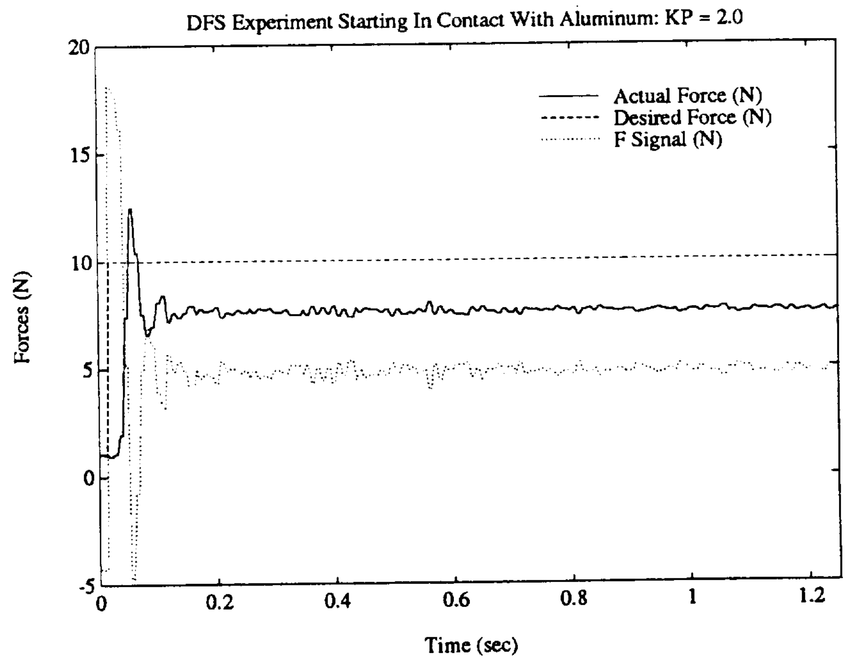


Figure 5.22: Proportional Gain Can Be Increased When Starting In Contact With Aluminum Surface

proportional force control increased when the surface being contacted was more flexible. This was opposite to the pattern of behaviour for the integral force control.

5.6 Summary

This chapter discussed a series of experiments designed to investigate how the flexibility of the environment may affect the stability of force control algorithms. The results seem to indicate that stiffer surfaces improve the range of stability for direct integral force control, while more compliant surfaces improve the stability range of direct proportional force control and position accommodation control. From the results presented here, it is clear that the environmental flexibility affects the force control parameters which can be used; therefore, any force control algorithm must take into account the type of surface to be contacted.

CHAPTER 6
Improvements to the Direct Force Servoing Algorithm

6.1 Motivation

As was shown in the previous chapters, both the PAC and DFS algorithms have difficulty when initially coming into contact with the environment. A typical example of the DFS algorithm is shown in Figure 6.1. The gripper was positioned about 1.5 mm above an aluminum block, and the DFS algorithm was initiated. As the plot shows, when the gripper first contacts the environment, there is a large force spike, after which the gripper quickly achieves the desired force on the environment.

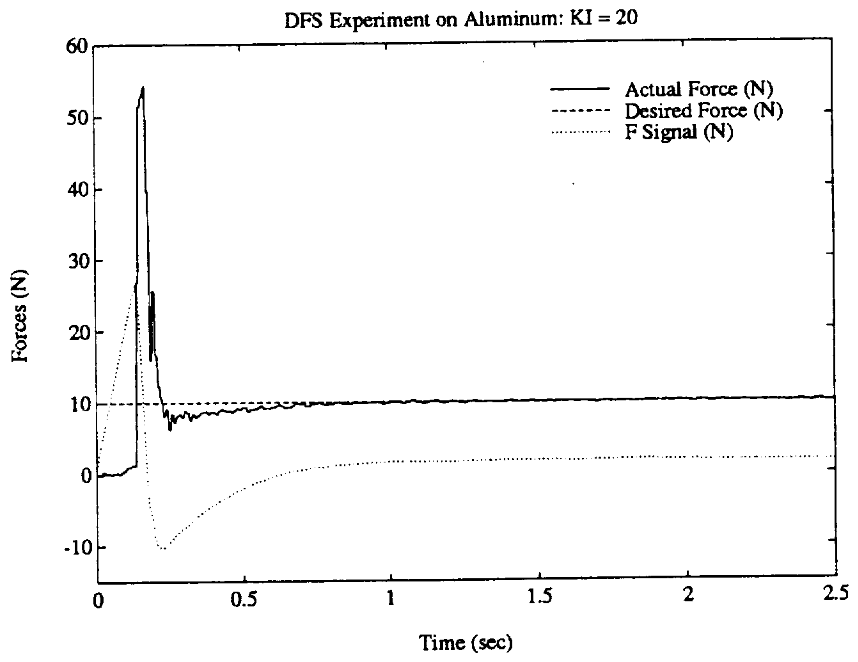


Figure 6.1: Typical Performance OF DFS on Aluminum

Figures 5.13–5.14 show the performance of the basic integral force feedback algorithm on wood for a range of integral gains. Note that as K_I is increased, the response of the system speeds up. This is desirable as a tighter loop implies better transient response (steady state force is quickly reached) and disturbance rejection (a disturbance force will cause smaller deviation from the setpoint — assuming that the arm stays in contact with the surface). However, higher gain also results in a larger force spike when the gripper first contacts the aluminum surface. For the case when $K_I = 30$, the gripper bounces once on the surface. This effect limits the practical size of the integral gains, even though higher gains can be used if the arm always stays in contact with the surface.

As noted in Chapter 5, the initial force spike increases as the integral gain is increased (for the same starting height above the surface). If this spike becomes large enough, it will cause the gripper to lift off the surface again due to the integral control. (This effect has also been noted by [10]). For practical operations, it is imperative to reduce the initial force spike when coming into contact with the environment.

An additional problem is illustrated by Figures 6.2 and 6.3. These figures show how the basic DFS Algorithm may behave when a disturbance is applied to the robot while it is in contact with the environment. In this experiment, a disturbance was applied at time $t = 6.75$ seconds. Because it was fairly large, the integrator in the force loop wound up; when the disturbance was removed, the arm overcompensated due to the integral term. Safety code then stopped the arm because the impact force exceeded a threshold. This example clearly shows that the system is not robust with respect to disturbances even though the arm behaves stably while in contact.

Motivated by these examples, modifications to the basic DFS algorithm should help to achieve the following goals:

- Reduce the force spikes upon contact with the environment.
- Maintain contact with the environment in the presence of disturbances.

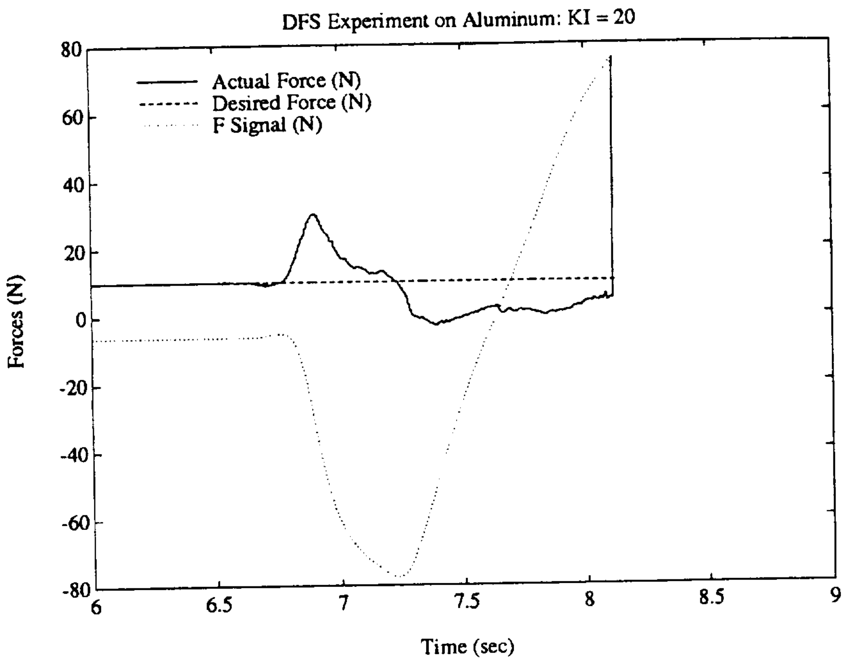


Figure 6.2: Disturbance Induced Instability in Basic DFS Algorithm: Force Profile

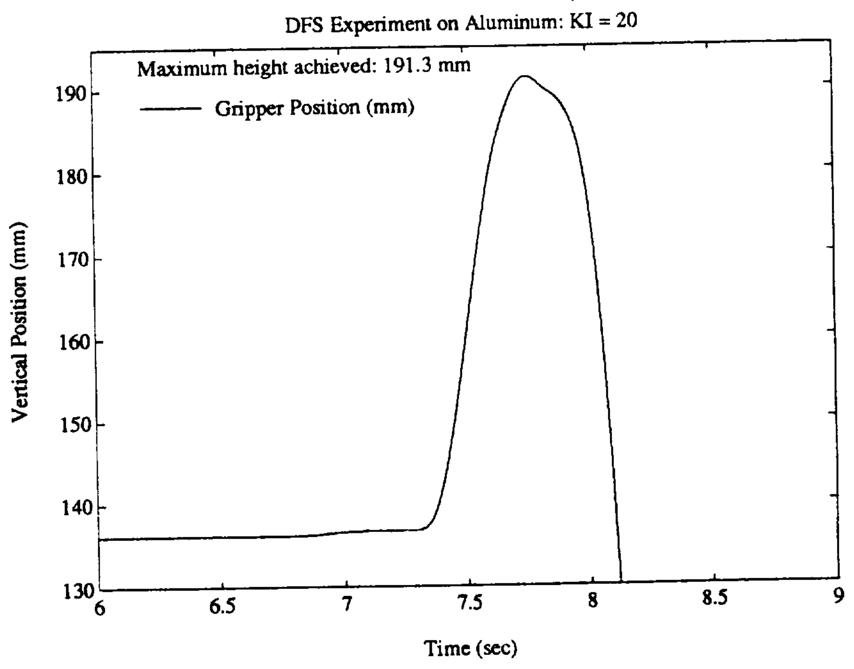


Figure 6.3: Disturbance Induced Instability in Basic DFS Algorithm: Position Profile

6.2 Modifications to the Basic DFS Algorithm

Three modifications to the basic DFS algorithm are discussed in this chapter. Test results are provided to demonstrate how each modification affects the performance of the DFS algorithm. Finally, all modifications are combined at the end of this chapter.

6.2.1 Integral Error Scaling

The control law in the basic algorithm (recall that K_P and K_{f_d} were set to 0) is of the form:

$$F = K_I \int_0^t f_{err}(s)ds \tag{6.1}$$

where $f_{err}(t) = (f_d - f(t))$. As was shown in Chapter 5, for a suitable range of K_I, this control law stabilizes the closed-loop system.

The integral error term will tend to cause overshoot in the force applied to the environment. It is important to realize that the robot–environment system is such that it is more acceptable to overshoot in one direction than the other. Overshoot when the robot is pushing into the surface (increasing the force) results in an actual force which is higher than the desired force. Overshoot when the robot is pulling away from the surface (decreasing the force) may result in the gripper breaking contact with the environment. In the process of reestablishing contact, there may be high impact forces which cause the system to go unstable.

Consider the following situation: The robot gripper is touching the environment, and applying a force of 10 Newtons. Assume a disturbance of 20 Newtons is applied so the actual force becomes 30 Newtons for a period of time. The integral action will try to force the arm off the surface in order to reduce the force back to the 10 Newton setpoint. If this response is too fast, the contact may be broken. If a disturbance of 20 Newtons is applied in the opposite direction, then the contact would be broken. It is now desirable to bring the arm back in contact with the surface quickly to reestablish contact. Thus, it is desirable to have different speeds of response depending on the direction of the force overshoot. One way to help achieve this behavior is to modify the error term f_{err} in the control law, based on the sign of the force error:

$$f_{err}(t) = \begin{cases} f_d - f(t) & \text{if } f_d - f(t) \geq 0 \\ \beta(f_d - f(t)) & \text{if } f_d - f(t) < 0 \end{cases} \tag{6.2}$$

where $0 < \beta \leq 1$.

Consider how this would affect the scenario described above.

When the initial force on the surface is 10 Newtons, and the 20 Newton disturbance pushes the gripper away from the surface, the controller will attempt to overcome the disturbance just like the unmodified algorithm. If the 20 Newton disturbance causes the actual force to increase above 10 Newtons, however, the f_{err} term is attenuated due to the scale factor, β. Thus, the integral error term doesn't change as quickly, and the tendency for the gripper to bounce back off the surface at this point is reduced. The result is a reduced retraction motion of the arm, which may either result in a smaller jump from the surface or eliminate the breakoff of contact altogether.

Figure 6.4 shows the effect of decreasing the scale factor β on the performance of the DFS algorithm. Note that as β is decreased, it takes longer for the robot to achieve the desired force. As β approaches 0, the response time of the system will approach infinity. That is, if $\beta = 0$, the system would overshoot the desired force, and would not decrease again, since the integral term in the control law would stop changing. Thus, there is a tradeoff between the overshoot that can be tolerated and the response time of the system.

Figure 6.5 shows the force trajectory using the DFS algorithm and integral error scaling. In this case, $\beta = 0.2$. Comparing this with Figure 6.1, notice that the

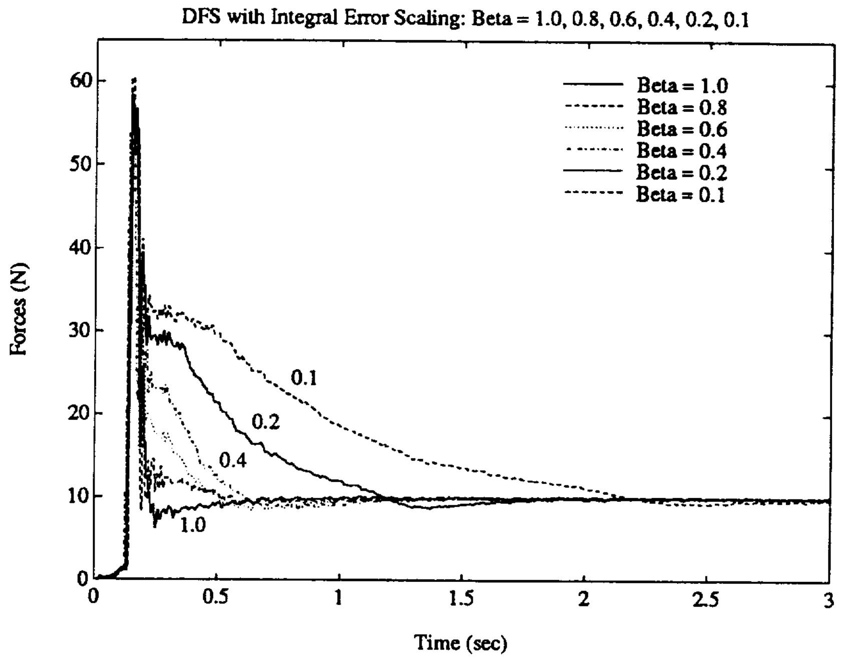


Figure 6.4: Effect of Scaling the Integral Error Term in DFS Algorithm

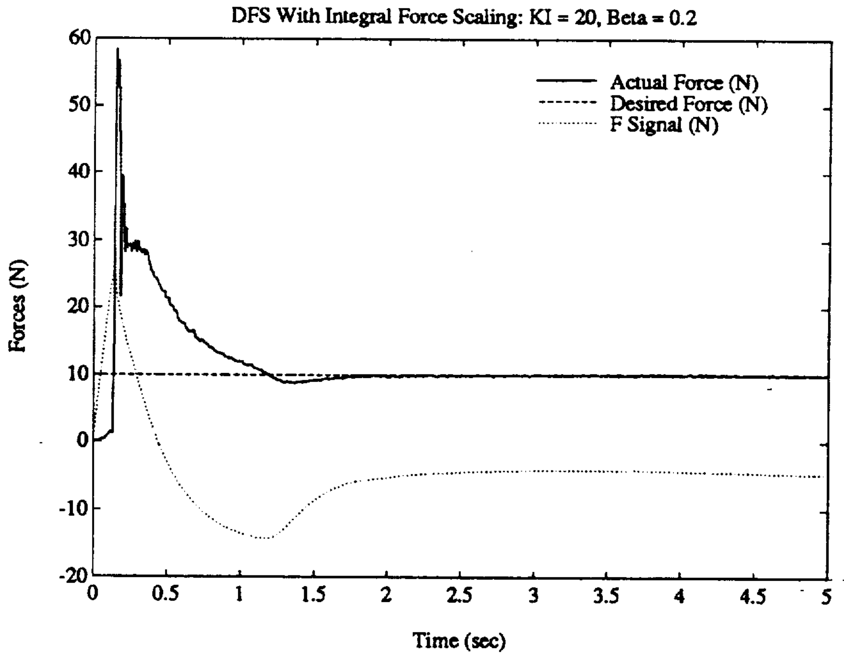


Figure 6.5: DFS Algorithm With Integral Error Scaling

force spike has not been reduced at all. However, the rate of change of the actual force (and the control signal) has been reduced.

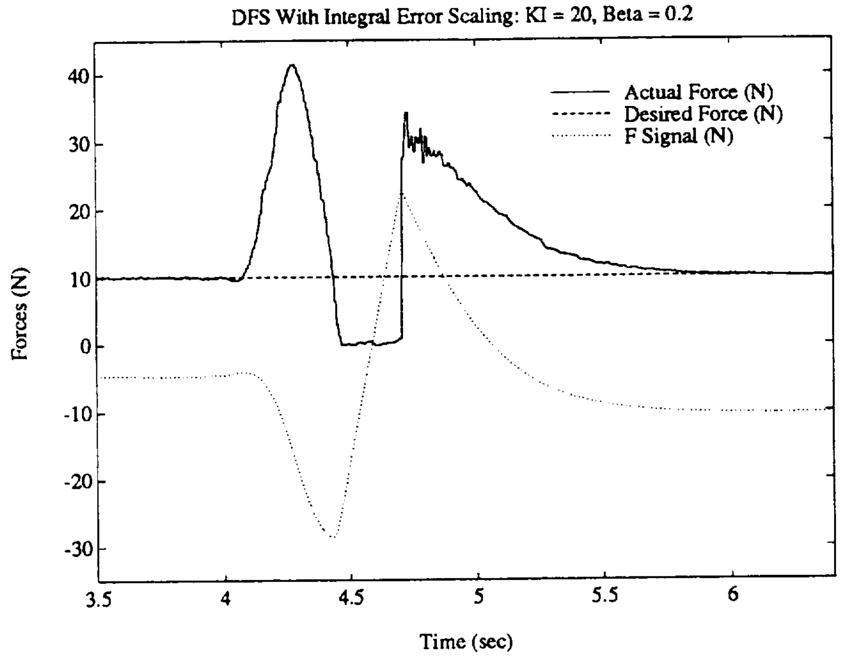


Figure 6.6: Disturbance Rejection of DFS Algorithm With Integral Error Scaling

Figure 6.6 shows the performance of the modified system when the robot was disturbed as before. Note that the disturbance is larger than in the initial case (Figure 6.2), but that the amplitude of the control signal is smaller, and that the reaction of the gripper is somewhat improved. The gripper came off the surface slightly, but returned without going unstable, and again settled on the desired force.

6.2.2 Variable Desired Force

A force control algorithm is used to allow the robot to come in contact safely with the environment. Thus, it is necessary to start a force control algorithm prior to coming in contact with the environment. As mentioned previously, the robot was positioned approximately 1.5 millimeters above the surface and the force control algorithm was used to initiate contact with the environment.

When the DFS algorithm is started, the integrator begins to wind up. This builds up the torque applied to the joints until the arm moves into the surface of the environment. However, between the time that the arm starts moving and the time that the gripper contacts the surface, the integrator winds up even further. This excessive windup contributes to an overshoot of the force applied to the surface (the initial force spike). Increasing the control gain K_I improves the response of the system, but adds to the windup and increases the impact force.

Consider the feedback control law in equation (6.1). If f_d is close to f initially, then f_{err} is small and the integrator will wind up more slowly than it would if f_{err} were large. Thus, another method to improve the behavior of the DFS algorithm is to modify the desired force so that it is "close" to the actual force initially, and after contact with the environment ensures that the final force value (in this case, 10 Newtons) is achieved. By doing this, the integrator windup should be reduced, thereby reducing the impact force.

While the gripper isn't in contact with the environment, the actual force is zero. It would be desirable to have a small positive desired force, so that the integrator winds up slowly. After contact, the desired force value should be increased to its final level.

Figure 6.7 shows the desired "trajectory" of the desired force. The plot requires some explanation. For the experiments presented here, the desired force value that was used was based upon the actual force reading. In this way, the desired force value could be kept "close" to the actual force reading, helping to reduce integral windup.

In order to prevent force spikes (due to impact, disturbances, or noise) from causing a spike in the desired force reading, the forces read from the sensor were filtered using a simple discrete–time first order filter:

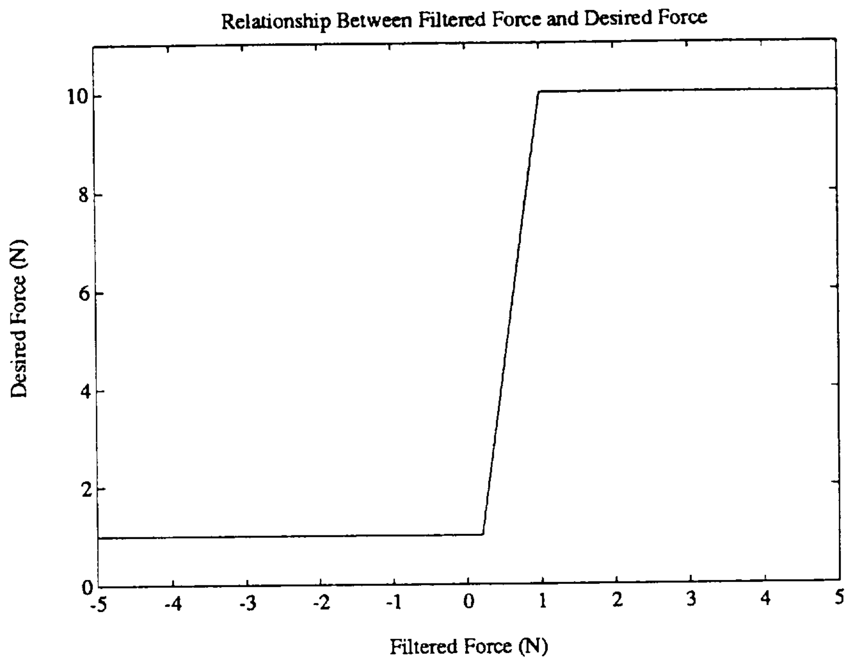


Figure 6.7: Modifying f_d Based on Filtered Force Readings

$$f_{filt}[k] = \alpha f_{filt}[k-1] + (1-\alpha)f[k], \quad 0 \leq \alpha \leq 1 \tag{6.3}$$

The f_d value at every sampling interval is calculated as follows: the force sensor is read, and the force value is filtered. This filtered force value is then used to calculate the desired force, f_d, according to the relationship depicted in Figure 6.7. This value of f_d is then used in the feedback control law, along with the *unfiltered* force, f. The filtered force is not used in the control law; it is only used in the determination of f_d.

It should be noted that the slope of the transition between the initial and final desired force values is important. In the transition, it is desirable to have $f_d(x) > f_{filt}(x)$. That way, for any given filtered force, the corresponding desired force will have a greater value. This will tend to increase the actual force, and so the actual force will tend toward the final desired force.

Figure 6.8 shows the performance of the basic DFS algorithm modified to

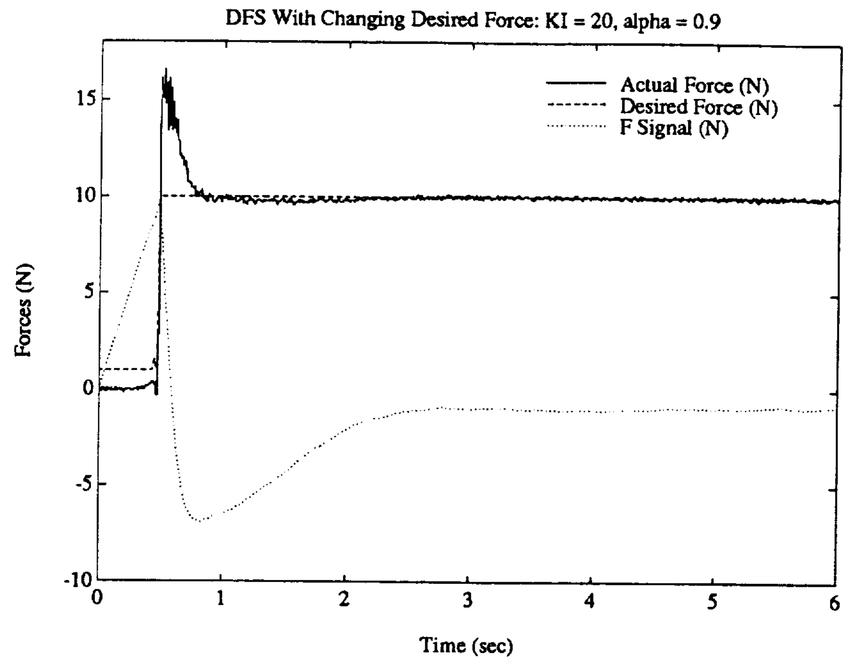


Figure 6.8: Effect of Modifying the Desired Force Based on Contact Force

vary the desired force as described above. Note that the desired force starts out at 1 Newton, and upon contact with the surface quickly ramps up to 10 Newtons. A comparison of Figures 6.8 and 6.1 shows some other important points. As predicted, the control signal is reduced when the desired force is modified. Further, notice that the initial force spike has been reduced considerably.

An important consideration in the implementation of this modification is the choice of the filter coefficient. Figure 6.9 shows the result when the filter coefficient is chosen too large (too close to 1). The filter becomes very low-pass, and the force control algorithm is unable to adjust f_d for rapidly changing forces. In the above experiment, the filter was so slow that the gripper settled on the initial desired force of 1 Newton, and after several seconds the desired force was increased to 10 Newtons, while the actual force tracked it.

Note that if α is set to 1, the goal force of 10 Newtons would never be achieved, since the filter would never take in any information about the actual force readings.

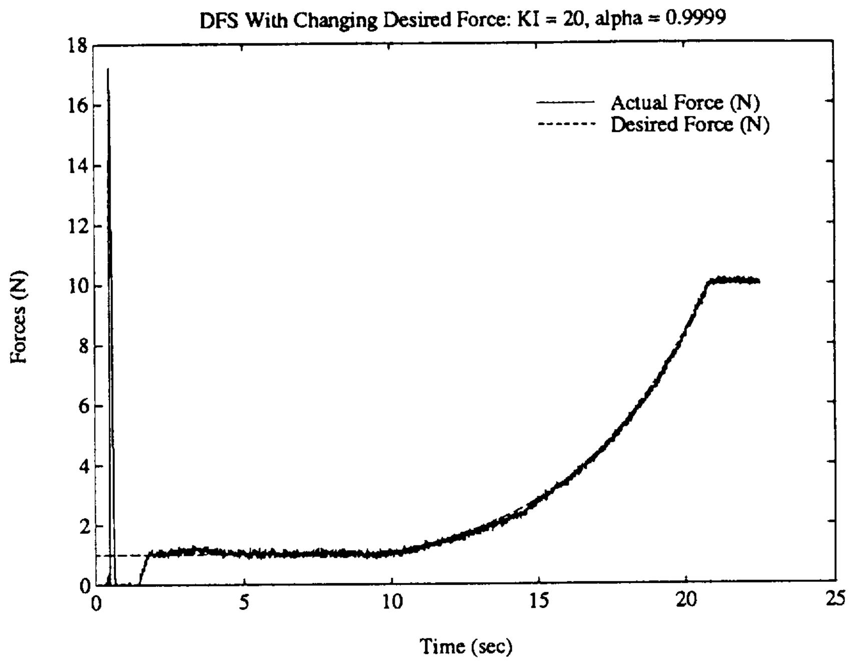


Figure 6.9: Result of Excessive Filtering on Desired Force Trajectory

If α is set to zero, the filter is not being used, and the desired force is then subject to noise spikes in the actual force readings. Thus, there is a tradeoff between the response time of the system and its susceptibility to force spikes.

Figure 6.10 shows the performance of the system in the presence of a disturbance in the actual force. The disturbance occurs at about $t = 5.6$ seconds. When the disturbance is removed from the system, the control signal causes the system to undershoot the desired force. In this case, the gripper leaves the surface of the aluminum block. Notice that the desired force value is reduced when the force readings go to zero, and return to the final value when the gripper contacts the surface again. Note also that the integrator winds up more slowly when the gripper loses contact with the surface than when the gripper is in contact with the surface.

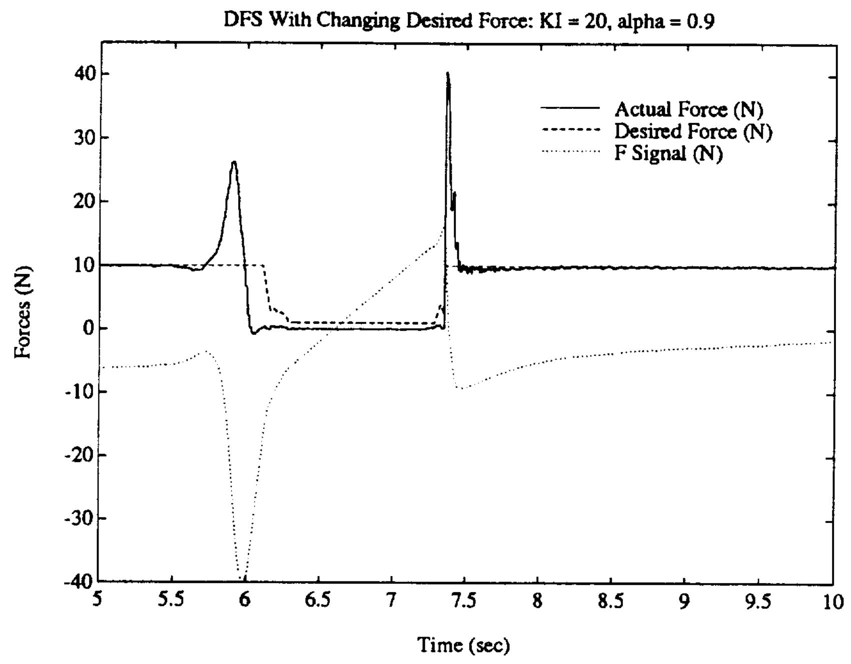


Figure 6.10: Disturbance Rejection of DFS Algorithm With Force Trajectory

6.2.3 Force Signal Clipping and Integral Windup Prevention

Another modification to the basic DFS algorithm is to clip the control signals which are calculated in the basic control routine. For the experiments presented here, two control signals were clipped:

- F — the control signal which is multiplied by the transpose Jacobian (J^T) in order to obtain the joint torques.
- The integral of the error term, $\int_0^t f_{err}(s)\,ds$.

The force control signal F was clipped because this limits the maximum torque which can be applied to the joints. By limiting F, the maximum force which can be applied to the environment is also limited. Care must be taken to ensure that the force signal can still be made large enough so that the robot can achieve the desired force.

The integral error was clipped so as to prevent the integrator from winding up too far. If a disturbance (such as shown in Figure 6.2) is too large, the integrator may wind up far enough to cause excessive forces and instability when the disturbance is removed.

Clipping signals is a common strategy in control systems. Indeed, all physical systems have physical limitations, and cannot output signals with infinite amplitude. This modification to the algorithm alters the control equations as follows:

$$\tau = J^T \operatorname{sat}(F) + \hat{g}(\theta) \tag{6.4}$$

$$F = K_I \operatorname{sat}\left(\int_0^t f_d - f(t)dt\right) \tag{6.5}$$

The saturation function, sat(), is defined as:

$$\operatorname{sat}(x) = \begin{cases} \rho_1 & x \leq \rho_1 \\ x & \rho_1 < x \leq \rho_2 \\ \rho_2 & x < \rho_2 \end{cases} \tag{6.6}$$

where $\rho_1 < \rho_2$.

Figure 6.11 shows the effect of this modification on the performance of the basic DFS algorithm. In this case, the control signal was clipped at ± 20 Newtons. The integral term was clipped at ± 2 Newtons. Since $K_I = 20$, this allowed the unclipped F signal to reach 40 Newtons. Notice that that in the beginning of the plot, (just prior to the force spike), the force control signal is clipped at 20 Newtons. A comparison of this figure with Figures 6.1 and 6.8 shows that the control signal F is less than the case when only the basic algorithm was used, but more than the case where the desired force was varied.

Figure 6.12 shows the disturbance rejection abilities of this modification to the DFS algorithm. Even in the presence of a prolonged large disturbance, the system

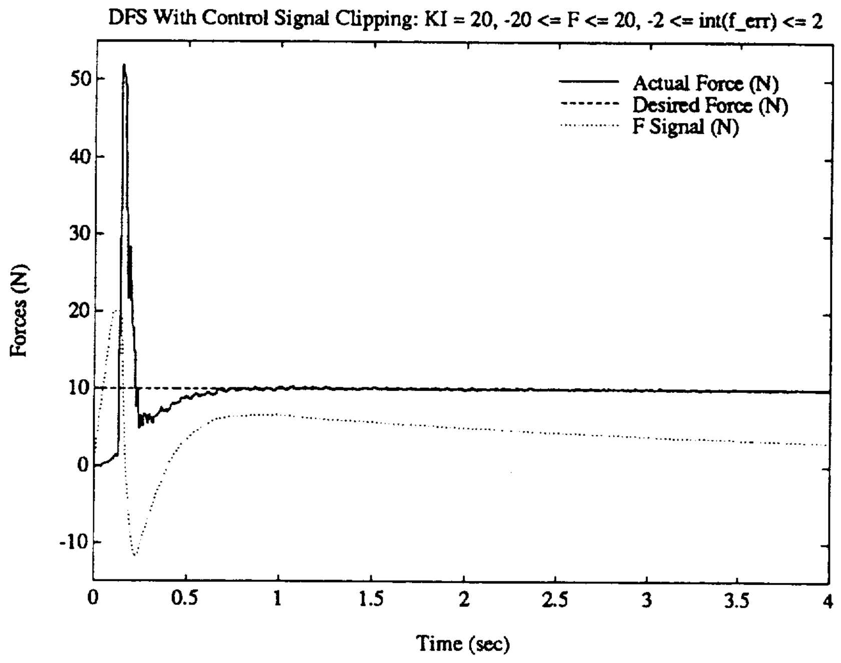


Figure 6.11: DFS Algorithm With Control Signal Clipping and Integrator Windup Prevention

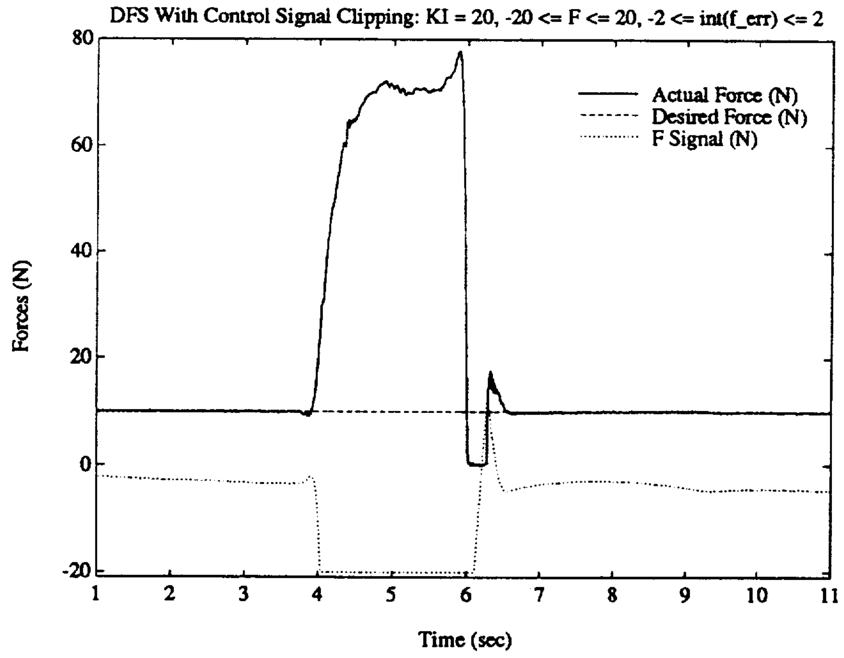


Figure 6.12: Disturbance Rejection of DFS Algorithm With Control Signal Clipping and Integrator Windup Prevention

was able to recover nicely and return to the desired force level.

6.2.4 Combined Modifications

Each of the previous modifications help the DFS algorithm in some way. The signal clipping limits the windup of the integrator, and prevents excessively large control signals. Varying the desired force also helps limit the force control signal, and provides a more natural "trajectory" for the force controller to follow than just a constant force setpoint. The scaling of the integral error term helps prevent the gripper from leaving the environment surface. By combining the modifications, an improved DFS algorithm is obtained. This algorithm has many of the best features from the above modifications.

The control equations for the augmented DFS algorithm then become:

$$\tau = J^T \operatorname{sat}(F) + \hat{g}(\theta) \tag{6.7}$$

$$F = K_P(f_d - f) + K_I \operatorname{sat}\left(\int_0^t f_{err}(s)ds \right) + K_{f_d} f_d(t) \tag{6.8}$$

$$f_{err}(t) = \begin{cases} f_d(t) - f(t) & \text{if } f_d(t) - f(t) \geq 0 \\ \beta(f_d(t) - f(t)) & \text{if } f_d(t) - f(t) < 0 \end{cases} \tag{6.9}$$

where $0 < \beta \leq 1$. The value of f_d is modified as described above, according to the relation shown in Figure 6.7.

Figures 6.13 and 6.14 show the position and force trajectories of the robot gripper. As before, the gripper was started at about 1.5 mm above the surface. Notice that the initial impact force is reduced due to the varying desired force modification. The robot was then disturbed by having a person push on the robot at time $t = 10$ seconds. Notice that the control torque ramped up to the maximum

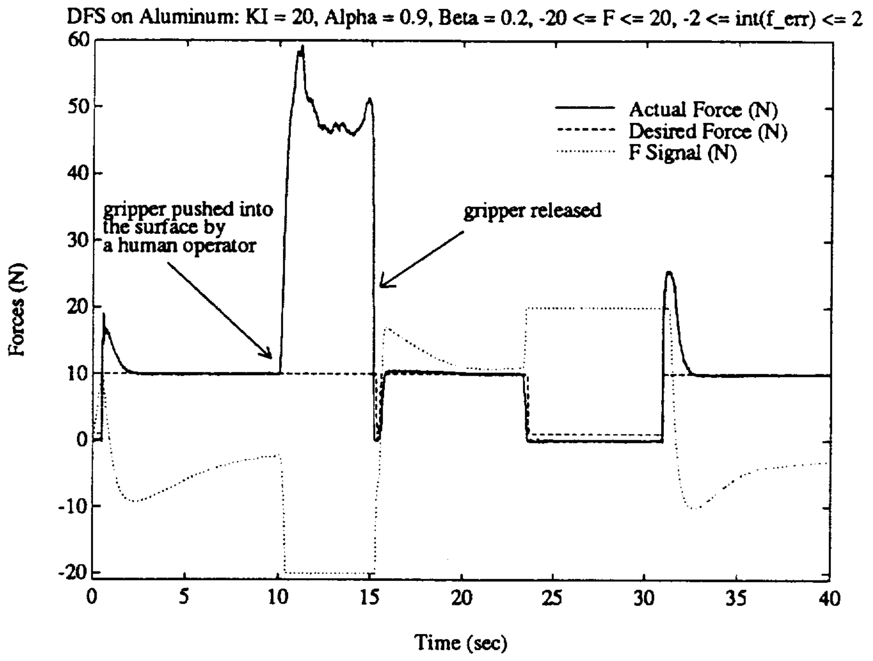


Figure 6.13: Disturbance Rejection of DFS Algorithm With All Improvements: Force Profile

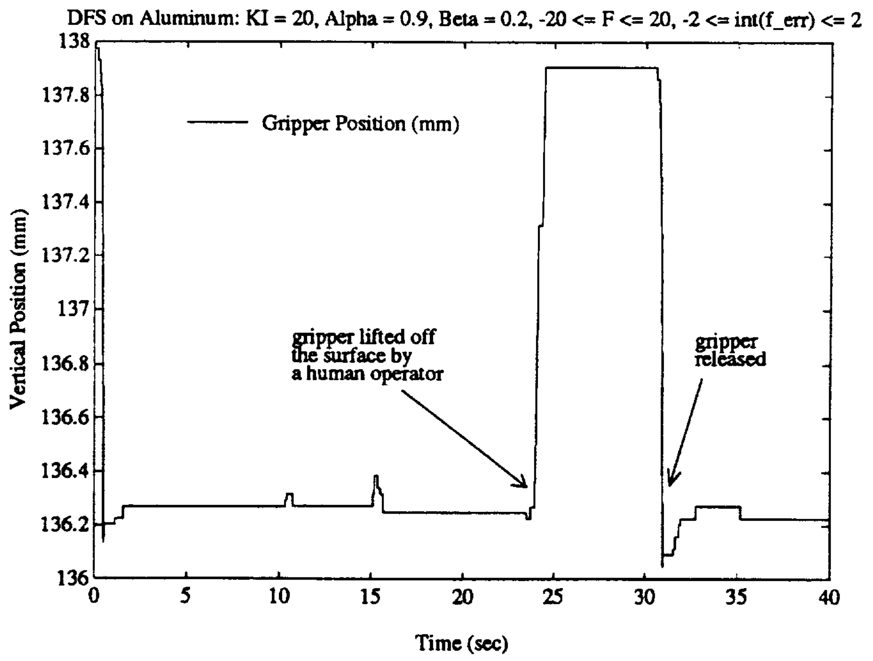


Figure 6.14: Disturbance Rejection of DFS Algorithm With All Improvements: Position Profile

and was then clipped. When the robot was released, the force returned to normal. The gripper only changed vertical position slightly (much less than a millimeter).

At time $t = 23$ seconds, a second disturbance was introduced. The gripper was lifted off the surface, allowing the controller to wind up in the other direction. Note that the desired force also changed, in an attempt to help limit the windup. When the robot was released, the robot again settled on the surface, and achieved the desired force of 10 Newtons.

6.3 Summary

In this chapter, three modifications to the Direct Force Servoing Algorithm have been presented. These modifications help to reduce the initial impact force of the robot gripper on the environment and maintain contact with the environment in the presence of disturbances. By combining all of the modifications into the DFS algorithm, the best features of each modification have been incorporated. The result is a force control algorithm which maintains contact with the surface and remains stable in the presence of large force disturbances.

CHAPTER 7
Further Discussion of Position Accommodation Control and Direct Force Servoing

7.1 Introduction

This chapter suggests general tuning processes for the two force control algorithms. The intent is to provide the reader with some insight into the tradeoffs and problems which are encountered when tuning each of the controllers. A comparison of the Direct Force Servoing and Position Accommodation Control algorithms is also presented, in order to provide the reader with the an idea of the benefits and costs of each method. The first section discusses the tuning process for each algorithm, while the second compares the two algorithms on a number of issues.

7.2 Choosing Gains In a Flexible Environment

The goal of the tuning process is to determine a set of gains which will allow the given force control algorithm to stably contact the environment, and apply some desired force. It would be preferable to have one set of gains which could be used to reliably contact a wide variety of different surfaces. The tuning processes are described below. Because the two algorithms are so different, the tuning procedures differ as well.

7.2.1 Position Accommodation Control

7.2.1.1 Mass

The results presented in Chapter 5 suggest several strategies for tuning the PAC force control algorithm. One of the most important is to have low mass or zero mass in the simulated MSD system. This allows the algorithm to alter the desired setpoint as quickly as possible in response to changes in the measured forces.

The theory presented in Chapter 3 indicates that elimination of the mass term also helps to eliminate oscillatory behaviour in the convergence of the forces. By removing the mass parameter, the order of the system is reduced, along with the configuration dependence.

7.2.1.2 Spring

In the Cartesian directions in which a given force is to be applied, there should be no spring force (that is, $K_d = 0.0$). As discussed in Chapter 3, the spring force serves to help keep the gripper close to the PAC algorithm's reference position. If the spring parameter is non–zero, the actual force in steady state will not equal the desired force.

For an axis (degree of freedom) in which no forces are to be applied, it may be reasonable to have a spring term. The spring term will tend to keep the gripper position constant along that axis. This helps prevent the gripper from drifting in other directions (due to noise in the force measurements) while the PAC algorithm is being used to apply a force along one axis. However, force disturbances along these other axes will cause the gripper to move out of the way, complying to the forces.

7.2.1.3 Damping

In the directions in which forces are to be applied, the damping parameter should be set to a conservative value (i.e. within the range of stability for the surface to be contacted). This value will determine the maximum speed of the gripper (recall that the maximum velocity will equal the force error divided by the damping). As the damping is increased, the velocity of the gripper will decrease, thereby decreasing the impact force. There is a tradeoff between how quickly the desired force can be achieved, and the impact force that can be tolerated.

For an axis in which no forces are to be applied, but in which compliance to external forces is desired, it is imperative that there be a damping term. The damping term is a dissipative element and helps to stabilize the system.

The tuning process essentially consists of setting the spring and mass terms to zero, and then lowering the damping parameter until a satisfactory balance between gripper velocity and impact force have been achieved. The data suggest that stiffer surfaces need the highest damping, especially when initially contacting the surface. Once tuned for stiffer surfaces, the same gains can be used for more compliant surfaces.

7.2.2 Direct Force Servoing

Both the theory and experimental results indicate that integral force control is more robust with respect to time delay than proportional force control. Integral control also is better in terms of steady state error. Thus, it is preferable to use integral force control instead of proportional control.

The experimental data suggest that K_I can be increased more for stiffer surfaces than for flexible surfaces. Thus, if the same control gains are to be used for several surfaces, it would be best to tune the controller for the most flexible surface.

The data also suggest that K_P can be increased more when the environment is flexible. Therefore, if proportional control is to be used to contact several surfaces, it would be best to tune the controller for the least flexible surface.

The tuning process for the basic DFS algorithm would consist of increasing the integral (proportional) gain until the desired response is achieved by the force controller. Increasing the gain will improve the response time of the controller, up to a point. As the gain is increased, oscillations in the force will set in, and eventually the system will go unstable.

Three modifications to the direct integral force algorithm were discussed in Chapter 6. These were designed to improve the robustness of the control algorithm. The improvements have additional parameters associated with them, and the adjustment of these parameters increases the complexity of the tuning process. They will be discussed below.

7.2.2.1 Integral Error Scaling

As discussed previously, the integral error scaling trades off the settling time of the system with the ability of the system to handle overshoot in the force measurements. In so doing, it helps the gripper to stay in contact with the environment.

The parameter which can be tuned for this modification is β, and is shown in equation (6.2). This parameter will directly affect the K_I gain. Consider the problem of increasing the integral gain: as the gain is increased, eventually the impact force causes the gripper to lift off the surface. The integral error scaling term (β) opposes this by slowing the response time when the actual force is greater than the desired force. By decreasing β, the response time is increased, allowing the integral gain to be increased.

Figure 7.1 shows two examples of the DFS algorithm being used to contact a piece of wood from a height of 1.5 millimeters. Table 5.4 shows that the largest stable K_I gain for the basic DFS algorithm in such a situation was 33.0. These two runs show that the β term can have an significant impact on the range of K_I.

Note that in the plot for $K_I = 100.0$, the gripper bounced twice on the surface. With $\beta = 0.2$, it was found that the integral gain could be increased to 102.0 before the system went unstable. By decreasing β to 0.1, however, it was possible to increase K_I still further. See Figure 7.2.

Note that modifying the parameter β does not affect the size of the impact force. It does however, improve the ability of the integral controller to handle large

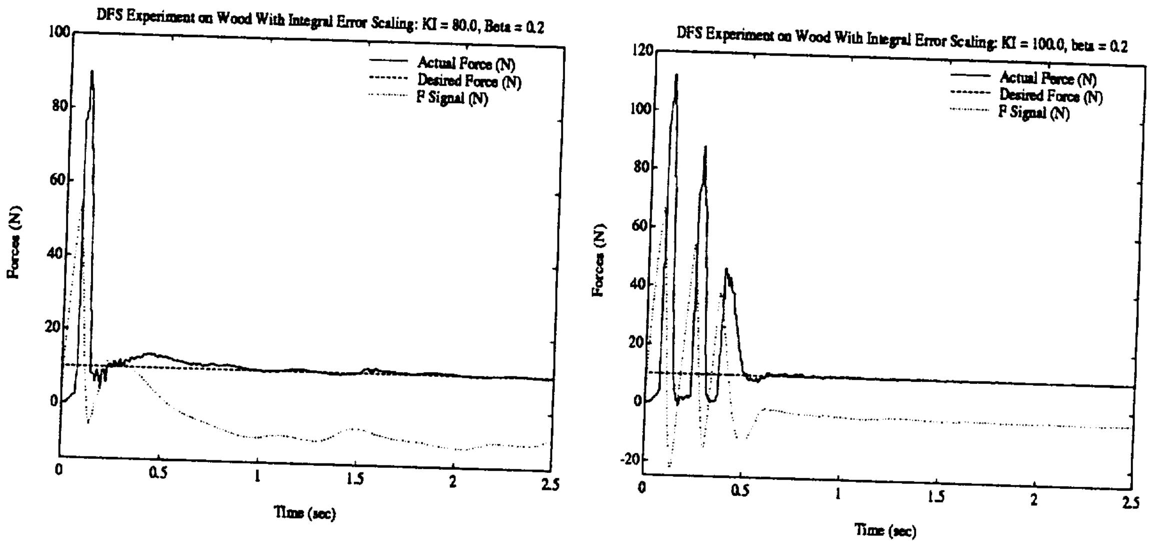


Figure 7.1: Integral Error Scaling Allows Higher Integral Gains

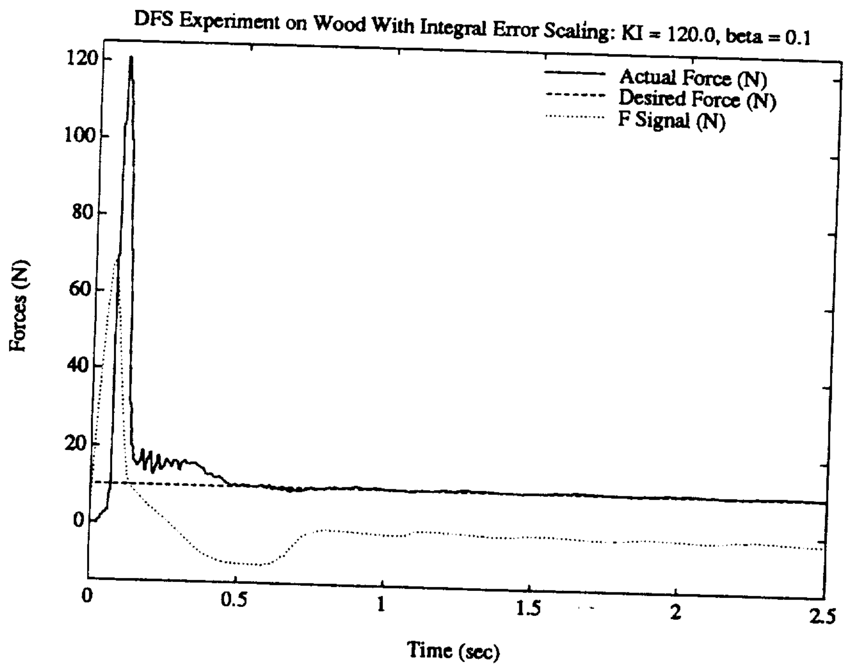


Figure 7.2: Still Higher Gains Can Be Achieved By Reducing Beta

force spikes.

The effect of β and K_I on the system should be kept in mind when tuning the controller. For a given integral gain, decreasing β will increase the settling time of the system. (Figure 6.4 illustrates this.) The settling time can be reduced again by increasing the value of K_I, but this in turn will cause larger impact forces.

The size of the impact force which can be tolerated by the environment will limit the size of the integral gain which can be used. In turn, the settling time which can be tolerated will limit the value of β.

7.2.2.2 Variable Desired Force

Varying the desired force helps to reduce the impact force when the gripper contacts the environment. The variable desired force modification actually has five parameters. Four of the parameters determine the shape of the force trajectory, by specifying the corner points of the piecewise linear relationship between the filtered and desired force. The fifth parameter is the coefficient for the filter which is used on the force measurements.

As discussed in Chapter 6, care must be taken when choosing the curve which describes the relationship between the filtered and desired force. Specifically, if the controller is being used to come into contact with the environment, it is important that the desired force be larger than the filtered force until the goal force is achieved. If this relationship holds, then the controller will tend to increase the force applied to the environment until the actual force reaches the goal.

The requirement that the desired force be larger than the filtered force can be pictured geometrically. Figure 7.3 shows a line where the desired force equals the filtered force. This is a line of equilibrium; an equilibrium point will exist where the filtered–desired force curve crosses this line. In order to have the desired force be larger than the filtered force, the filtered–desired force curve must lie *above* this equilibrium line.

Those sections of the filtered–desired force curve which lie above this line will cause the controller to increase the force applied to the environment, because the desired force will be larger than the actual force readings. The sections of the curve which are below the equilibrium line will cause the controller to decrease the force applied to the environment because the desired force will be less than the actual force measurements.

The corner points of the filtered–desired force curve should be chosen so that the transition (from non–contact desired force to the in–contact desired force) is above the equilibrium line. This will allow the controller to achieve the final in–contact desired force, by making the in–contact desired force an equilibrium point. The filtered–desired force curve which was used in the experiments shown in Chapter 6 is depicted in Figure 7.3. Note that the transition in the curve lies above the line of equilibrium. It crosses the equilibrium line at (10,10); thus, the equilibrium point which the controller will try to achieve is a force of 10 Newtons.

Figure 7.3 also shows an example of a poor filtered–desired force relationship. Note that one of the corner points of the curve was chosen below the equilibrium line, so that the curve actually crosses the equilibrium line at around 5 Newtons. In this case, 10 Newtons is not a stable equilibrium point; the force will tend to settle around the equilibrium point of 5 Newtons.

Figure 7.4 shows the performance of the DFS algorithm using this relationship. Notice that the desired force changes from 1 Newton to 10 Newtons initially (due to the impact force), and then gradually decreases to 5 Newtons. The actual force follows; as predicted, the resulting steady state force value is 5 Newtons.

The slope of the transition should be chosen based on how fast the desired force should change from one level to the other after contact is made with the surface. If a slower change is desired, a more gradual slope can be specified.

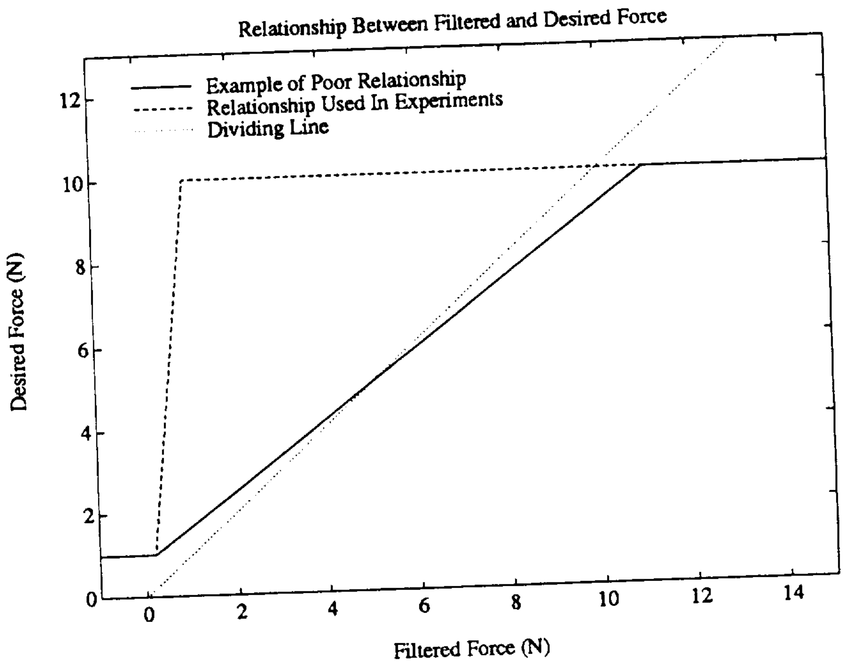


Figure 7.3: Relationship Between Filtered and Desired Force Must be Chosen Carefully

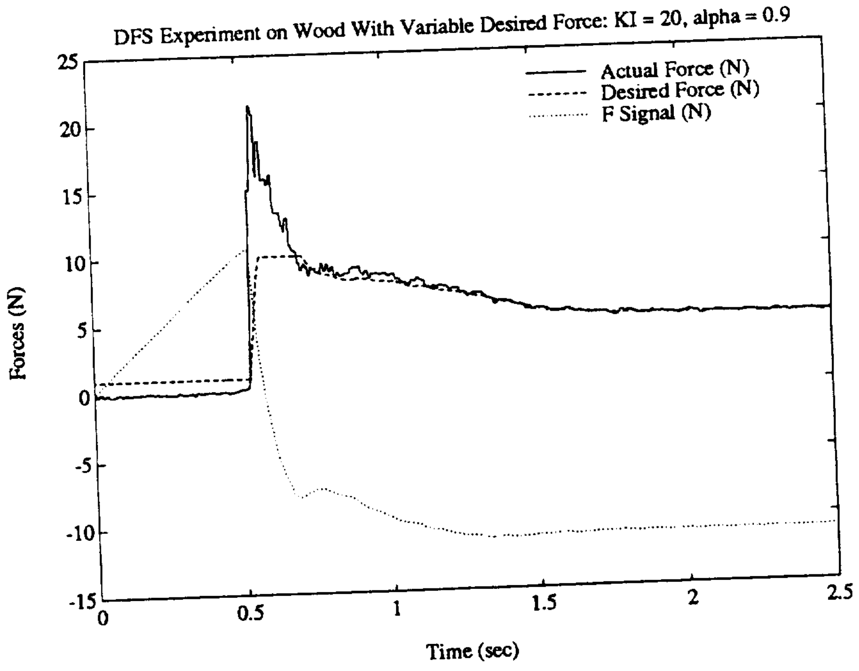


Figure 7.4: Effect of Poor Relationship Between Filtered and Desired Force

In addition to being placed above the line of equilibrium, the lower corner point of the filtered–desired force curve has another constraint. The filtered–force coordinate of this corner point should be chosen above the noise level of the force sensor. This helps to prevent the desired force from changing until the robot has actually contacted the environment. For example, the force sensor used in the experiments provided readings which would fluctuate by an amount less than 0.2 Newtons. The lower corner point of the filtered–desired force curve was set at (0.2, 1.0); this meant that sensor noise would not cause the desired force to increase. After the gripper contacted the environment, the measured force was no longer zero. The filtered force then increased, causing the desired force to increase as well.

The filter coefficient α trades off between noise rejection and sensitivity to fast changes in the force measurements. This parameter also helps to prevent noise from altering the desired force before the gripper contacts the environment. Tuning this parameter should be done by considering the noise in the system and how fast the forces are likely to change (given disturbances, etc.). For many of the experiments which were performed, a value of $\alpha = 0.9$ was found to provide reasonable noise rejection, while still allowing the controller to alter the desired force quickly in the presence of a disturbance.

7.2.2.3 Force Signal Clipping and Integral Windup Prevention

This modification to the DFS algorithm worked by limiting two different control signals; the integral error and the force control signal (which was multiplied by the transpose Jacobian).

Each signal that was clipped required two parameters in order to specify the limits; the upper limit and the lower limit on the signal. Having the two limits specified individually allowed the controller to apply a greater force in one direction than the other. For simplicity, however, the magnitude of the upper and lower limits was set to be the same. For example, if the upper limit of a signal was $+2$, the lower limit was set to -2.

A number of problems may be encountered with these parameters. If the robot arm has any joint friction and the limit of F is set too low, the arm may not move, although the integral error will wind up completely. Another possible occurrence is that the gripper is able to contact the surface, but is unable to achieve the desired force. In both cases, the controller cannot output a signal large enough to overcome joint friction and achieve the desired force.

For the experiments which were run, the limit for the force signal F was set roughly 2–3 times larger than the desired force value. This allowed the controller to put out enough torque to overcome joint friction and achieve the desired force.

The limit on the integral error also must be set. If this limit is set too low, the maximum F signal (determined by the limits on F) may never be achieved. If the limit is set too high, the integrator will be able to wind up too far.

To determine the limits on the integral error, the limits of F were used, along with the value of K_I. As a starting point, the integral error was limited to a value which would generate an unclipped F signal twice the value of the limited F signal. This was done to allow the integrator to wind up, giving the controller some "memory", without allowing it to wind up too far. As an example, if $K_I = 20.0$, and the limits on F were ± 20.0 N, then the integral error limits would be set at ± 2.0.

7.3 Qualitative Comparison of Force Control Algorithms

In addition to exploring the effect of environmental flexibility, the experiments discussed in Chapter 5 were also structured in such a way as to allow some qualitative comparison of the two force control algorithms. For example, the robot was placed in a similar configuration for all test runs. In addition, the sampling rates were selected in such a way that both algorithms received new force information at the same rate.

The PAC and the DFS algorithms are fundamentally different; they approach the force control problem in different ways. The Position Accommodation algorithm is an indirect method of force control. This means that the forces applied to the environment are controlled by controlling the position of the end–effector of the robot. In contrast to this approach, the Direct Force Servoing algorithm is a direct method of force control; forces applied to the environment are controlled by specifying the motor torques applied to the joints of the robot.

The PAC algorithm is a higher–level force control, being located in the trajectory generator. Because of this, the algorithm must be run at the servo rate of the trajectory generator. This is typically an order of magnitude slower than the joint controller rate. In addition, the PAC algorithm depends upon the lower–level joint controller to handle the dynamics of the robot arm. The actual mass of the robot is ignored by the force control algorithm; instead the end–effector is made to mimic the dynamics of a simple mass–spring–damper system. It has been shown experimentally that the choice of the underlying joint controller can drastically affect the performance of the joint controller [20].

In comparison, the DFS algorithm is a lower–level approach to the force control problem. Because it is located in the joint controller of the robot, it has no lower control layer to depend on. Because it runs at the controller servo rate, it is reasonable to expect that the DFS algorithm would be more responsive than the PAC algorithm in typical robotic systems.

Because of the locations of the algorithms in the robot control architecture, the PAC algorithm is inherently better suited for industrial controllers. Such controllers are typically position–based, and it is relatively easy to build a trajectory generator to supply the position controller with setpoints. The DFS algorithm, on the other hand, requires a specialized joint controller that can handle force measurements.

Another major difference between the algorithms presented here is that the Position Accommodation algorithm moves the end-effector in Cartesian space. This is easy for users to visualize, and can help when attempting to tune the controller to apply forces to objects in the workspace of the robot. The Direct Force algorithm, as implemented here, does not move the gripper in straight lines in Cartesian space. This is a relatively unattractive feature of this control method. However, the algorithm can be modified in order to correct this problem, by adding a Cartesian motion controller. Equation (4.1) would be altered to become:

$$\tau = J^T(F_{force} + F_{pos}) + \hat{g}(\theta) \tag{7.1}$$

In the above equation, F_{force} is the force control signal due to the force error measured at the end-effector. Thus it is the same as F, used elsewhere in this document. F_{pos} is the force control signal due to the Cartesian position error of the end-effector, and would be generated using a PD or PID feedback control law. Typically, the gripper position would be controlled along some Cartesian axes, while forces applied by the gripper would be controlled along the remaining axes.

Still another difference between the algorithms is exhibited by their behaviour around robot singularities. As discussed in Chapter 3, the PAC algorithm has severe difficulties with robot singularities, due to its use of inverse kinematics. Near the singular points of the robot, small Cartesian motions may result in large joint motions.

In contrast to the behaviour of the PAC algorithm, the DFS algorithm is very well behaved near singular points. This is because the heart of the DFS algorithm lies in the use of the transpose Jacobian. At singular points, the Jacobian matrix will become singular, but the implication of this is only that the robot can no longer exert forces in the degree of freedom that was lost. No large joint motions will occur, even though the robot loses a degree of freedom.

PAC	DFS
• indirect method	• direct method
• Cartesian space	• not Cartesian space
• located in trajectory generator	• located in joint controller
• depends on underlying controller	• no underlying joint controller
• good for industrial controllers	• need specialized controller
• problem with singularities	• little problem with singularities

Table 7.1: Summary of Comparison of PAC and DFS algorithms

Table 7.1 summarizes the comparison between the different force algorithms. The key comparisons that have been presented in this section have been highlighted.

7.4 Summary

This chapter discussed a rough tuning procedure for both the PAC and DFS force control algorithms. Tuning the PAC algorithm mainly consisted of adjusting the damping parameter. The basic DFS algorithm only required adjustment of the integral gain; however, the addition of the improvements to the DFS algorithm added extra complexity to the tuning process.

A qualitative comparison of both algorithms was also presented and discussed in this chapter. Because the two algorithms are fundamentally different, there are a number of contrasts between them.

CHAPTER 8
Conclusions and Future Research

8.1 Summary and Conclusions

This report presented the theory and implementation of two different types of force control algorithms. These algorithms were tested on several different surfaces to determine the effect of environmental flexibility on the stability of the control algorithms. The experimental results from these experiments have been presented and discussed. Suggestions for improving the DFS algorithm were also presented, and verified with experimental results. Finally, a suggested tuning process for each control algorithm was provided.

A number of things can be concluded from the results presented in this report.

- The flexibility of the environment has a significant impact on the stability of the force control algorithm. A force control algorithm must take into account the type of surface being contacted.

- Force spikes due to impact are a problem which a force control algorithm must be able to deal with.

- Integral force control is better than proportional force control for achieving a desired force. Integral control has zero steady–state error for constant desired force setpoints, and is more robust than proportional force control.

- Increasing the integral gains (higher K_I for DFS, lower damping for PAC) improves the response time of the control algorithm, but also increases the impact forces which occur when the robot contacts the environment.

- Direct integral force control is not robust with respect to disturbances. However, modifications to the basic algorithm can be made which improve the robustness of the algorithm. These modifications increase the complexity of the tuning process.

8.2 Future Research

The experimental results presented in this report have shown several areas in which more research is necessary. These topics are discussed below.

8.2.1 Friction Identification and Compensation

The robot arms at CIRSSE have significant static and viscous friction. It is known that static friction causes limit cycles when a PID algorithm is used to control joint positions. This phenomenon is easy to recognize: the robot joints will oscillate slowly about their setpoints. This is caused by friction in the joints preventing the motion of the joint when there is a slight error in the location of the joint position. The integrator in the PID control winds up, and when the output torque from the controller is sufficient to overcome friction and move the joint, the joint overshoots the desired setpoint. The process then repeats, in reverse. The result is a slow, small-amplitude oscillation about the setpoint. PD control also has problems with friction. In this case, friction prevents the joint from ever reaching the desired setpoint.

In order to improve the positioning capability of the joint controllers, it is desirable to be able to compensate for friction in some way. One avenue of research is to estimate the friction parameters of each of the robot arm joints. The parameters would include static, viscous, and Coulomb friction. Once the parameters have been estimated, it should be straightforward to adjust the control torques to compensate for the friction parameters.

Another possibility is to implement an adaptive friction compensation algorithm. Such an algorithm would adjust the friction compensation torque in real-time in order to counteract the effects of friction. For such an algorithm, it might not be necessary to explicitly estimate the friction parameters of the arm joints.

8.2.2 Estimation of Environmental Flexibility

The range of control gains for which the robot-environment system remained stable depended upon the flexibility of the surface being contacted. This result highlights the need to be able to estimate the environmental flexibility of the environment. If a robotic system can estimate the flexibility of the environment, the force control gains could then be adjusted automatically to perform better for that surface.

8.2.3 Cartesian Motion and DFS

The DFS method outlined in this report is not a Cartesian-space controller. (Refer to Section 4.1.2.) A near-term research goal is to build a Cartesian controller which uses a similar J^T approach. One advantage to such a controller is that the task space is easier for the user to visualize. It is also much easier to build a hybrid force and position control system when working in Cartesian space.

8.2.4 Hybrid Force and Position Control

Hybrid force and position control is an important issue related to Cartesian DFS control. Hybrid control allows the robot to control forces in some directions, and positions in others.

Such a control capability is very important if a robot is to perform meaningful tasks. If the robot were to write something on a blackboard, for example, it must control the force applied to the blackboard, or the chalk may break. However, it must also control the position of the chalk in the directions parallel to the surface of the blackboard.

8.2.5 Active biasing of force sensor

One of the built-in functions of the Lord force sensor is that of force sensor "biasing". Essentially, it is a zeroing out of the current force readings. Every subsequent force reading is then modified by having the bias reading subtracted from it. This functionality is very useful; if the robot is holding a 5 Newton weight in its gripper, for example, it may be convenient to consider the 5 Newton force the "zero force" reading.

The biasing function of the Lord sensor has a severe limitation, however. The biasing only works if the sensor is not rotated about an axis perpendicular to the direction of gravity. Thus, if the sensor is rotated about any horizontal axis, the sensor will register forces due to its own weight (and the weight of the gripper). If the force sensor is rotated about a vertical axis, no change to the biasing will take place.

An interesting avenue of research is in "active biasing" of the force sensor. In this case, several force measurements are taken, with the force sensor in different orientations. The biasing code would then use these force readings as reference points in order to bias all subsequent force readings properly, regardless of the orientation of the force sensor. It should be noted that the biasing code would need to be supplied with the orientation of the sensor in order to function correctly.

8.2.6 Velocity observer

The interface to the PUMAs at CIRSSE has a severe limitation in that only position values can be read from the Unimate Controllers. Since PID and PD control require the velocity value as well, it is necessary to either estimate the velocity with an observer, or to take the derivative of the position signal.

Currently, a derivative method is employed in the software-based joint controllers. Since this method is inherently noisy, a filter must be used to help smooth out the velocity signal. The resulting signal is still noisy, and is not the true velocity of the joints. Additionally, there is a tradeoff which must be considered; as the bandwidth of the filter is lowered, the filter pole interferes more with the controller poles. While tuning the joint controllers for the PUMA, it was necessary to consider the filter as a part of the controller, increasing the complexity of the pole placement process.

The alternative method of using a velocity observer seems more promising. The observer is expected to produce smoother and more accurate velocity signals, without the need for filtering. This should result in better arm control and an easier tuning process.

8.2.7 Two Arm Force Control

The work presented here lays the groundwork for research in two-arm control. The CIRSSE testbed consists of two 9 DOF arms like the one shown in Figures 2.3 and 2.4. Preliminary two–arm control has already been achieved at CIRSSE [20]. There are a number of difficulties in two–arm control; both arms must comply to each other as well as to external forces. As with single–arm control, the arms must also come into contact with the environment in order to perform useful tasks. It is expected that the work presented here will be directly applicable to this research.

LITERATURE CITED

[1] D. Whitney, "Historical perspective and state of the art in robot force control," *International Journal of Robotic Research*, vol. 6, no. 1, pp. 3–14, 1987.

[2] N. Hogan, "Impedance control: An approach to manipulation, parts I-III," in *ASME Journal on Dynamic Systems, Measurement and Control*, vol. 107, pp. 1–24, Mar. 1987.

[3] M. Raibert and J. Craig, "Hybrid position/force control of manipulators," *ASME Journal on Dynamic Systems, Measurement and Control*, vol. 102, pp. 126–133, June 1981.

[4] O. Khatib and J. Burdick, "Motion and force control of robot manipulators," in *Proc. 1986 IEEE Conference on Robotics and Automation*, (San Francisco, CA), pp. 1381-1386, Mar. 1986.

[5] C. An and J. Hollerbach, "Dynamic stability issues in force control of manipulators," in *Proc. 1987 IEEE Conference on Robotics and Automation*, pp. 890–896, 1987.

[6] D. Wedel and G. Saridis, "An experiment in hybrid position/force control a six DOF revolute manipulator," in *Proc. 1988 IEEE Conference on Robotics and Automation*, (Philadelphia, PA), pp. 1638–1642, 1988.

[7] J. Wen and S. Murphy, "Robot force control in the presence of environmental flexibilities," in *Sixth Yale Workshop on Adaptive and Learning Systems*, (New Haven, CT), pp. 225–230, Aug. 1990.

[8] J. Wen and S. Murphy, "Stability analysis of position and force control for robotic arms," *IEEE Transaction on Automatic Control*, vol. 36, pp. 365–371, Mar. 1991.

[9] R. Volpe, *Real and Artificial Forces in the Control of Manipulators: Theory and Experiments*. PhD thesis, Carnegie Mellon University, 1990.

[10] R. Volpe and P. Khosla, "Experimental verification of a strategy for impact control," in *Proc. 1991 IEEE International Conference on Robotics and Automation*, (Sacramento, CA), pp. 1854-1860, Apr. 1991.

[11] R. Volpe and P. Khosla, "An experimental evaluation and comparison of explicit force control strategies for robotic manipulators," in *Proc. 1992 American Control Conference*, (Chicago, IL), pp. 758-764, 1992.

[12] J. F. Watson, III, "Testbed kinematic frames and routines," CIRSSE Technical Memorandum 1 (v. 2), Rensselaer Polytechnic Institute, Troy, NY, August 1991.

[13] J. J. Craig, *Introduction to Robotics, Mechanics & Control.* Reading, Mass.: Addison-Wesley, 1986.

[14] Lord Corporation (Industrial Automation Division), Cary, NC, *Installation and Operations Manual for F/T Series Force/Torque Sensing Systems*, revision 3.1 ed., November 1987.

[15] Wind River Systems, Incorporated, Alameda, CA, *VxWorks Programmers's Guide*, 1990.

[16] K. R. Fieldhouse, K. Holt, D. R. Lefebvre, S. H. Murphy, D. Swift, and J. F. Watson, III, "Lecture materials for the CTOS/MCS introductory course," CIRSSE Report 97, Rensselaer Polytechnic Institute, Troy, NY, August 1991.

[17] B. C. Kuo, *Automatic Control Systems.* Englewood Cliffs, New Jersey: Prentice–Hall, Inc., fifth ed., 1987.

[18] L. S. Wilfinger, "7 DOF gravity compensation for the testbed arms," CIRSSE Technical Memorandum 15 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, February 1992.

[19] K. Holt and L. Wilfinger, "Jacobian library for the testbed arms," CIRSSE Technical Memorandum 17 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, July 1992.

[20] M. Ryan, S. Murphy, and J. Wen, "Force regulation in multiple manipulator systems," in *Fourth Annual CIRSSE Conference*, (Troy, NY), Oct. 1992.

[21] D. Sood, "Kinematics and jacobian of the platform and the robot." Internal communication, 1992.

[22] A. Fijany and A. Bejczy, "Efficient jacobian inversion for the control of simple robot manipulators," engineering memorandum, Jet Propulsion Laboratory, April 1988.

[23] B. Armstrong, O. Khatib, and J. Burdick, "The explicit dynamic model and inertial parameters of the PUMA 560 arm," in *Proc. 1986 IEEE Conference on Robotics and Automation*, (San Francisco, CA), pp. 510–518, Mar. 1986.

[24] M. Leahy, L. Nugent, and G. Saridis, "Efficient PUMA manipulator jacobian calculation and inversion," *Journal on Robotic Systems*, 1987.

[25] C.-J. Li, A. Hemani, and T. Sankar, "An efficient computational method of the jacobian for robot manipulators," *Robotica*, vol. 9, pp. 231–234, 1991.

[26] S. Murphy and D. Swift, "Dynamic parameters and inverse dynamics for the PUMA 560," CIRSSE Technical Memorandum 13 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, January 1992.

[27] T. J. Tarn, A. K. Bejczy, H. Shuotiao, and X. Yun, "Inertia parameters of PUMA 560 robot arm," Robotics Laboratory Report SSM-RL-85-01, Department of Systems Science and Mathematics, Washington University, September 1985.

[28] D. Swift, "Kinematic and dynamic parameters for the testbed grippers and loads," CIRSSE Technical Memorandum 14 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, January 1992.

APPENDIX A
Derivation of Gravity Compensation Equations

This section discusses the derivation of the gravity compensation equations for the PUMA arms as they are mounted on the CIRSSE testbed. Because of the extra degrees of freedom that the CIRSSE testbed possesses, the equations presented in works such as [23] were inadequate.

A.1 Notation and Conventions

The notation in this chapter is consistent with that outlined in Chapter 1. In addition, there are a number of other conventions which are used in this chapter. For completeness, the relevant conventions are summarized below.

- The coordinate frames of the CIRSSE testbed arm are labeled 1 – 9; frame 0 is the global origin (see [12]). An E denotes the end–effector frame.

- Due to the design of the CIRSSE testbed arm, the joints of the PUMA are labeled 4 – 9. The subscript labels of parameters reflect this labeling as well. Thus, link 6 of the PUMA is considered link 9 of the arm, and the mass of this link would me labeled m_9.

- ${}^kg$ is the vector which describes the acceleration due to gravity, expressed in the coordinates of frame k. When g is expressed in frame 0, the prescript will be dropped.

$${}^0g = g = \begin{bmatrix} 0 & 0 & -9.8062 \end{bmatrix}^T \frac{m}{s^2}$$

- Estimated values will be specified with a hat symbol over it. Thus, an estimate of the acceleration due to gravity would be designated as $\hat{g}$, while the actual value would be specified as g.

- $-{}^k\tau_i$ is the 3-vector describing the link torque (due to gravity) felt at joint i, expressed in the coordinates of frame k. ${}^k\tau_i$ is the link torque applied to the joint by the motor.

- ${}^k r_{i,j}$ is the vector describing the position of the center of mass of link j with respect to frame i, and expressed in the coordinates of frame k.

$${}^k r_{i,j} = \begin{bmatrix} {}^k r_{i,j}(x) & {}^k r_{i,j}(y) & {}^k r_{i,j}(z) \end{bmatrix}^T$$

  Note that under many circumstances, $k = i$. In this case, the i subscript may be dropped; thus ${}^k r_{k,j} = {}^k r_j$.

- ${}^k p_{i,j}$ is the 3×1 vector describing the position of frame j with respect to frame i, expressed in the coordinates of frame k. Under many circumstances, $k = i$. In this case, the i subscript will be dropped; thus, ${}^k p_{k,j} = {}^k p_j$. Note also that ${}^k p_{i,j} = -{}^k p_{j,i}$.

- ${}^i_j R$ is the 3×3 rotation matrix describing the orientation of frame j with respect to frame i.

- ${}^i_j T$ is the 4×4 homogeneous transformation describing the position and orientation of frame j with respect to frame i. Thus:

$${}^i_j T = \begin{bmatrix} {}^i_j R & {}^i p_j \\ 0 & 1 \end{bmatrix}$$

- ${}^k \tilde{p}_{i,j}$ is the 3×3 cross product matrix associated with the vector ${}^k p_{i,j}$, expressed in the coordinates of frame k:

$${}^k \tilde{p}_{i,j} = \begin{bmatrix} 0 & -{}^k p_{i,j}(z) & {}^k p_{i,j}(y) \\ {}^k p_{i,j}(z) & 0 & -{}^k p_{i,j}(x) \\ -{}^k p_{i,j}(y) & {}^k p_{i,j}(x) & 0 \end{bmatrix}$$

The arguments x, y, and z in the above matrix represent the three components of the vector ${}^{k}p_{i,j}$. The term "cross product matrix" indicates that for any 3×1 vector ${}^{k}w$, the following equation holds:

$${}^{k}\tilde{p}_{i,j}\,{}^{k}w = {}^{k}p_{i,j} \times {}^{k}w$$

It should be noted that the following equation also holds for cross product matrices: ${}^{l}_{k}R\,{}^{k}\tilde{p}_{i,j}\,{}^{k}_{l}R = {}^{l}\tilde{p}_{i,j}$

- ${}^{k}\dot{x}_{i,j}$ is the velocity of frame j with respect to frame i, expressed in the coordinates of frame k. This is a 6×1 vector; the first three components of this vector are the linear velocity and the last three are the angular velocity:

$${}^{k}\dot{x}_{i,j} = \begin{bmatrix} {}^{k}\dot{x}_{L} \\ {}^{k}\omega \end{bmatrix}$$

  Note that under many circumstances, $k = i$. In this case, the i subscript will be dropped; thus ${}^{k}\dot{x}_{k,j} = {}^{k}\dot{x}_{j}$.

- ${}^{k}J_{i,j}$ is the Jacobian relating joint velocities of the robot to the velocity of frame j with respect to frame i. This relative velocity is expressed in the coordinates of frame k. Frame j will be referred to as the "velocity frame" since the joint velocities are mapped to the velocity of frame j. Frame i will be called the "velocity reference frame" (or "reference frame") since the velocity of frame j is measured with respect to frame i. Frame k will be referred to as the "coordinate frame."

- Trigonometric functions may be abbreviated by their first letter; therefore $C_i = \cos\theta_i$, and $S_i = \sin\theta_i$. Note that $C_{ij} = \cos(\theta_i + \theta_j)$.

A.2 High Level Equations

The calculation of the gravity compensation equations presented here was done by starting at the highest numbered links and working backwards. This was done because the weight of the upper links affects the torque felt at the joints of the lower links.

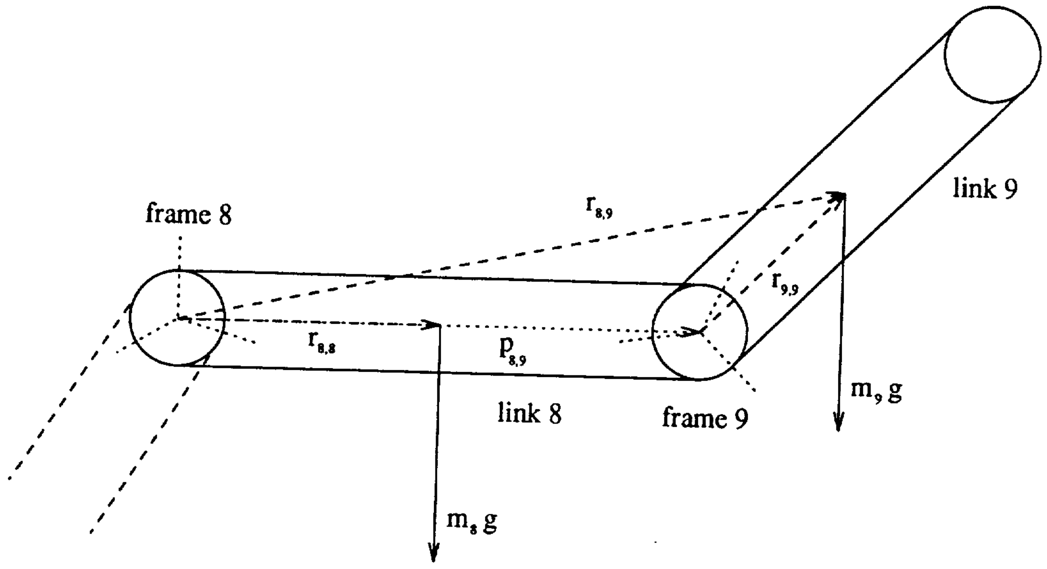


Figure A.1: Last Two Links of Robot Arm

Figure A.1 shows a representation of links 8 and 9 of the robot arm. The equation for the torque applied to joint 9, due to the mass of link 9 is:

$$-\tau_9 = r_{9,9} \times m_9 g$$

Expressed in frame 0, we have:

$$-{}^0\tau_9 = {}^0r_{9,9} \times m_9 g = {}^0_9R\,{}^9r_{9,9} \times m_9 g \tag{A.1}$$

The torque on joint 8 is due to the weights of both link 8 and link 9. Thus:

$$-{}^0\tau_8 = {}^0r_{8,8} \times m_8 g + {}^0r_{8,9} \times m_9 g$$

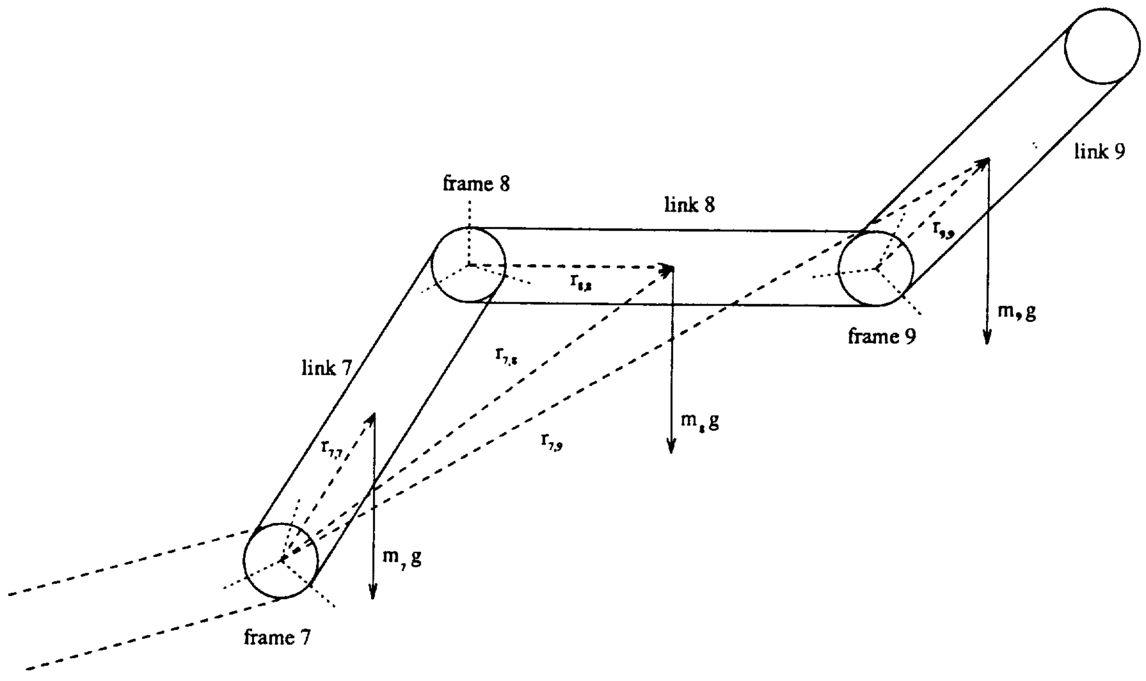


Figure A.2: Last Three Links of Robot Arm

Note that ${}^0r_{8,9} = {}^0_8R({}^8p_{8,9} + {}^8_9R\,{}^9r_{9,9})$. Substituting this into the above equation and expanding, we obtain:

$$-{}^0\tau_8 = {}^0_8R\,{}^8r_{8,8} \times m_8 g + {}^0_8R\,{}^8p_{8,9} \times m_9 g + {}^0_9R\,{}^9r_{9,9} \times m_9 g$$

$$-\,{}^0\tau_8 = {}^0_8R\,{}^8r_{8,8} \times m_8 g + {}^0_8R\,{}^8p_{8,9} \times m_9 g - {}^0\tau_9 \qquad \text{(A.2)}$$

See Figure A.2. This diagram shows that the torque on joint 7 is due to the weight of links 7, 8 and 9. The equation is:

$$-{}^0\tau_7 = {}^0r_{7,7} \times m_7 g + {}^0r_{7,8} \times m_8 g + {}^0r_{7,9} \times m_9 g$$

Expanding as before, and substituting terms, we obtain:

$$-\,{}^0\tau_7 = {}^0_7R\,{}^7r_{7,7} \times m_7 g + {}^0_7R\,{}^7p_{7,8} \times (m_8 + m_9)g - {}^0\tau_8 \qquad \text{(A.3)}$$

This procedure can be performed for all of the arm joints. For the upper six joints (the PUMA arm) the torque equations are:

$$\begin{aligned}
-{}^0\tau_9 &= {}^0_9R\,{}^9r_{9,9} \times m_9 g \\
-{}^0\tau_8 &= {}^0_8R\,{}^8r_{8,8} \times m_8 g + {}^0_8R\,{}^8p_{8,9} \times m_9 g - {}^0\tau_9 \\
-{}^0\tau_7 &= {}^0_7R\,{}^7r_{7,7} \times m_7 g + {}^0_7R\,{}^7p_{7,8} \times (m_8 + m_9) g - {}^0\tau_8 \\
-{}^0\tau_6 &= {}^0_6R\,{}^6r_{6,6} \times m_6 g + {}^0_6R\,{}^6p_{6,7} \times (m_7 + m_8 + m_9) g - {}^0\tau_7 \\
-{}^0\tau_5 &= {}^0_5R\,{}^5r_{5,5} \times m_5 g + {}^0_5R\,{}^5p_{5,6} \times (m_6 + m_7 + m_8 + m_9) g - {}^0\tau_6 \\
-{}^0\tau_4 &= {}^0_4R\,{}^4r_{4,4} \times m_4 g + {}^0_4R\,{}^4p_{4,5} \times (m_5 + m_6 + m_7 + m_8 + m_9) g - {}^0\tau_5
\end{aligned} \tag{A.4}$$

The equations in (A.4) define the link torques caused *by* gravity *on* the joints. To compensate for this torque, an opposing torque must be applied to the link by the joint motors. By negating the equations, the equations for the link compensation torques are obtained.

These 6 equations are *vector* equations which describe the torque on each of the joints, expressed in the coordinates of frame *0*. Each vector equation can be separated into 3 equations (one for each of the X,Y, and Z components of the vector). By expressing each torque τ_i in the i^{th} frame, the expressions for the torque vectors can be simplified. This is because the Z axis of the i^{th} frame is attached to the i^{th} link of the robot, and coincident with the i^{th} joint axis [13]. Since the i^{th} joint can only move about the i^{th} frame's Z axis, the only component of the vector ${}^i\tau_i$ that is of concern is the Z component.

Negating the equations, and applying the appropriate rotation matrices to (A.4), we obtain:

$$\begin{aligned}
{}^9\tau_9 &= {}^9_0R\,{}^0\tau_9 = & -{}^9_0R[{}^0_9R\,{}^9r_{9,9} \times m_9 g] \\
{}^8\tau_8 &= {}^8_0R\,{}^0\tau_8 = & -{}^8_0R[{}^0_8R\,{}^8r_{8,8} \times m_8 g + {}^0_8R\,{}^8p_{8,9} \times m_9 g - {}^0\tau_9] \\
{}^7\tau_7 &= {}^7_0R\,{}^0\tau_7 = & -{}^7_0R[{}^0_7R\,{}^7r_{7,7} \times m_7 g + {}^0_7R\,{}^7p_{7,8} \times (m_8 + m_9) g - {}^0\tau_8]
\end{aligned} \tag{A.5}$$

$$
\begin{aligned}
{}^{6}\tau_6 &= {}^{6}_{0}R\,{}^{0}\tau_6 = -{}^{6}_{0}R[{}^{0}_{6}R\,{}^{6}r_{6,6} \times m_6 g + {}^{0}_{6}R\,{}^{6}p_{6,7} \times (m_7+m_8+m_9)g - {}^{0}\tau_7] \\
{}^{5}\tau_5 &= {}^{5}_{0}R\,{}^{0}\tau_5 = -{}^{5}_{0}R[{}^{0}_{5}R\,{}^{5}r_{5,5} \times m_5 g + {}^{0}_{5}R\,{}^{5}p_{5,6} \times (m_6+m_7+m_8+m_9)g - {}^{0}\tau_6] \\
{}^{4}\tau_4 &= {}^{4}_{0}R\,{}^{0}\tau_4 = -{}^{4}_{0}R[{}^{0}_{4}R\,{}^{4}r_{4,4} \times m_4 g + {}^{0}_{4}R\,{}^{4}p_{4,5} \times (m_5+m_6+m_7+m_8+m_9)g - {}^{0}\tau_5]
\end{aligned}
$$

Finally, by expanding the equations and simplifying we obtain:

$$
\begin{aligned}
{}^{9}\tau_9 &= -[{}^{9}r_{9,9} \times m_9\,{}^{9}_{0}Rg] \\
{}^{8}\tau_8 &= -[{}^{8}r_{8,8} \times m_8\,{}^{8}_{0}Rg + {}^{8}p_{8,9} \times m_9\,{}^{8}_{0}Rg - {}^{8}_{9}R\,{}^{9}\tau_9] \\
{}^{7}\tau_7 &= -[{}^{7}r_{7,7} \times m_7\,{}^{7}_{0}Rg + {}^{7}p_{7,8} \times (m_8+m_9)\,{}^{7}_{0}Rg - {}^{7}_{8}R\,{}^{8}\tau_8] \\
{}^{6}\tau_6 &= -[{}^{6}r_{6,6} \times m_6\,{}^{6}_{0}Rg + {}^{6}p_{6,7} \times (m_7+m_8+m_9)\,{}^{6}_{0}Rg - {}^{6}_{7}R\,{}^{7}\tau_7] \\
{}^{5}\tau_5 &= -[{}^{5}r_{5,5} \times m_5\,{}^{5}_{0}Rg + {}^{5}p_{5,6} \times (m_6+m_7+m_8+m_9)\,{}^{5}_{0}Rg - {}^{5}_{6}R\,{}^{6}\tau_6] \\
{}^{4}\tau_4 &= -[{}^{4}r_{4,4} \times m_4\,{}^{4}_{0}Rg + {}^{4}p_{4,5} \times (m_5+m_6+m_7+m_8+m_9)\,{}^{4}_{0}Rg - {}^{4}_{5}R\,{}^{5}\tau_5]
\end{aligned}
\tag{A.6}
$$

The above equations (A.6) are the gravity compensation equations for the PUMA. The equations determine the (link) torques which must be applied *by the motors* in order to compensate for gravity.

A.3 Scalar Form of Gravity Compensation Equations

With the high level equations defined, we can now find the scalar equations for gravity compensation. From [13] we know that the general form of a homogeneous transformation describing the position and orientation of the i^{th} frame with respect to the $i-1^{th}$ frame is:

$$
{}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

The modified Denavit-Hartenberg parameters for the testbed can be found in [12]. With this information, we can build all of the needed homogeneous transformations. Substituting these transforms into equations (A.6), we obtain 6 vector equations, where each vector has 3 components, for a total of 18 equations. However, since all of the torque vectors are expressed in their link frame coordinates, only the Z component of each vector equation is of interest.

Arm	Link Number, i	mass (kg)	${}^ir_i(x)$ (m)	${}^ir_i(y)$ (m)	${}^ir_i(z)$ (m)
Right Arm	Link 4	12.95	0.0	0.0389	-0.3088
	Link 5	17.40	0.068	0.006	-0.016
	Link 6	4.80	0.0	-0.070	0.014
	Link 7	0.82	0.0	0.0	-0.019
	Link 8	0.34	0.0	0.0	0.0
	Link 9	0.09	0.0	0.0	0.032

Table A.1: Masses and Centers of Gravity of the PUMA Arm Links

These 6 equations are the scalar gravity compensation equations. They are not shown here due to their length. In order to simplify the equations, we can eliminate those terms which are zero. The masses of the arm links and the centers of gravity of the links were obtained from [23, 26, 27]. After eliminating zero terms and simplifying, we obtain the following equations:

$$\begin{aligned}
{}^9\tau_9 &= 0 \qquad\qquad (A.7)\\
{}^8\tau_8 &= C_3[K_1(C_{56}S_8 + S_{56}C_7C_8)] - S_3[K_1(S_4S_7C_8 - C_4C_{56}C_7C_8 + C_4S_{56}S_8)]\\
{}^7\tau_7 &= -C_3[K_1(S_{56}S_7S_8)] - S_3[K_1(C_4C_{56}S_7S_8 + S_4C_7S_8)]\\
{}^6\tau_6 &= C_3[K_1(C_{56}C_7S_8 + S_{56}C_8) - K_2C_{56} + K_3S_{56}] +\\
&\quad S_3[K_1(C_4C_{56}C_8 - C_4S_{56}C_7S_8) + K_2C_4S_{56} + K_3C_4C_{56}]\\
{}^5\tau_5 &= C_3[K_1(C_{56}C_7S_8 + S_{56}C_8) - K_2C_{56} + K_3S_{56} - K_4C_5 + K_5S_5] +\\
&\quad S_3[K_1(C_4C_{56}C_8 - C_4S_{56}C_7S_8) + K_2C_4S_{56} + K_3C_4C_{56} + C_4(K_4S_5 + K_5C_5)]\\
{}^4\tau_4 &= -S_3[S_4(K_1(C_{56}C_7S_8 + S_{56}C_8) - K_2C_{56} + K_3S_{56} - K_4C_5 + K_5S_5) +\\
&\quad C_4(K_1S_7S_8 - K_6)]
\end{aligned}$$

The constants K_1 - K_6 are defined as follows:

$$
\begin{aligned}
K_1 &= -m_9\,{}^9r_{9,9}(z)g \qquad (A.8)\\
K_2 &= (m_7 + m_8 + m_9)a_6 g\\
K_3 &= (m_6\,{}^6r_{6,6}(y) - m_7\,{}^7r_{7,7}(z) - (m_7 + m_8 + m_9)d_7)g\\
K_4 &= ((m_6 + m_7 + m_8 + m_9)a_5 + m_5\,{}^5r_{5,5}(x))g\\
K_5 &= m_5\,{}^5r_{5,5}(y)g\\
K_6 &= (m_5 d_5 + (m_6 + m_7 + m_8 + m_9)(d_5 + d_6) +\\
&\quad (m_4\,{}^4y_{4,4}(y) + m_5\,{}^5r_{5,5}(z) + m_6\,{}^6r_{6,6}(z)))g
\end{aligned}
$$

Equations (A.7) and (A.8) are the gravity compensation equations for the PUMA arms on the testbed. No terms have been discarded. It should be reiterated that these equations produce joint torques; it is necessary to scale the torques resulting from these equations by the gear ratios for the PUMA in order to determine the correct motor torques to be sent out to the Unimate controller.

A.4 Gripper and Load

The equations in the previous section calculate the compensation torque for the PUMA only; no provision is made for having a gripper attached to the end of the arm, or for a load being carried in the gripper.

However, the gripper (and any load in the gripper) can be modeled by an increase in mass of link 9 of the arm, along with a change in the center of gravity of link 9. Given the following terms:

- mass of link 9, m_9
- gripper mass, $m_{gripper}$
- load mass, m_{load}

- Z component of ${}^{9}r_{9,9}$, ${}^{9}r_{9,9}(z)$
- distance from frame 9 to the flange surface in Z direction, ${}^{9}p_{9,f}(z)$
- center of gravity of the gripper in the Z direction with respect to the flange, ${}^{9}r_{f,gripper}(z)$
- center of gravity of the load in the Z direction with respect to the flange, ${}^{9}r_{f,load}(z)$

the mass and center of gravity of the augmented link 9 is calculated as follows.

$$m_9' = m_9 + m_{gripper} + m_{load} \tag{A.9}$$

$$ {}^{9}r_{9,9}'(z) = \frac{m_9\,{}^{9}r_{9,9}(z) + m_{gripper}({}^{9}p_{9,f}(z) + {}^{9}r_{f,gripper}(z)) + m_{load}({}^{9}p_{9,f}(z) + {}^{9}r_{f,load}(z))}{m_9 + m_{gripper} + m_{load}} \tag{A.10}$$

Note that all of the distance terms in the above equations are expressed in the coordinates of frame 9. The parameters for the grippers mounted on the testbed arms can be found in [28].

A.5 Comparison With Other Work

Armstrong, et. al. [23] presents a set of equations for the dynamics of the PUMA arm. However, these equations assume a level PUMA. This assumption is inappropriate for the PUMAs of the CIRSSE testbed; the equations presented here allow for the PUMA to be tilted about a single axis. Thus, the equations derived here work well for the PUMAs as they are mounted on the platforms presently; they do *not* allow for the PUMA base to be tilted arbitrarily.

It should also be noted that by simplifying equations (A.6) to obtain (A.7), some limitations were introduced. By substituting 0 in for the terms ${}^{9}r_{9,9}(x)$ and

${}^{9}r_{9,9}(y)$, for example, the equations were greatly simplified, but at the expense of requiring that the center of gravity of the gripper (and of the load) be along the Z axis of frame 9. The equations given in [23] also have this limitation.

It can be easily shown that the equations in [23] are a simple case of the equations presented here. It should be noted that the first two joints of the 9 DOF robot arm have no effect on the gravity compensation equations. This is intuitive, given the geometry of the testbed arms; joint 1 is a level translational joint, and joint 2 is a rotational joint which turns about an axis parallel to the gravity vector. If $\theta_3 = 0$, then $\sin\theta_3 = 0$, $\cos\theta_3 = 1$, and equations (A.7) and (A.8) become:

$$\begin{aligned}
{}^{9}\tau_9 &= 0 \\
{}^{8}\tau_8 &= K_1(C_{56}S_8 + S_{56}C_7C_8) \\
{}^{7}\tau_7 &= -K_1(S_{56}S_7S_8) \\
{}^{6}\tau_6 &= K_1(C_{56}C_7S_8 + S_{56}C_8) - K_2C_{56} + K_3S_{56} + \\
{}^{5}\tau_5 &= K_1(C_{56}C_7S_8 + S_{56}C_8) - K_2C_{56} + K_3S_{56} - K_4C_5 + K_5S_5 \\
{}^{4}\tau_4 &= 0
\end{aligned} \tag{A.11}$$

$$\begin{aligned}
K_1 &= -m_9\,{}^{9}r_{9,9}(z)g \\
K_2 &= (m_7 + m_8 + m_9)a_6g \\
K_3 &= (m_6y_6 - m_7\,{}^{7}r_{7,7}(z) - (m_7 + m_8 + m_9)d_7)g \\
K_4 &= ((m_6 + m_7 + m_8 + m_9)a_5 + m_5\,{}^{5}r_{5,5}(x))g \\
K_5 &= m_5\,{}^{5}r_{5,5}(y)g
\end{aligned} \tag{A.12}$$

These equations and constants are identical with those given in [23]. Note that K_6 is not needed in this case.

APPENDIX B
Derivation of Computationally Efficient PUMA Jacobian Equations

B.1 Notation and Conventions

The notation in this chapter is consistent with that outlined in Chapter 1 and Appendix A. The reader is directed to these chapters for a summary of the conventions.

B.2 Background

The Jacobian matrix (or *Jacobian*) is a mapping between joint space and task (Cartesian) space. It maps joint velocities to Cartesian velocities (linear and angular) according to the following relationship:

$$\dot{x} = J\dot{q} \tag{B.1}$$

where $\dot{x}$ is a 6×1 vector of Cartesian velocities, J is a $6 \times n$ matrix, and $\dot{q}$ is a $n \times 1$ vector of joint velocities. (For the PUMA, $n = 6$.)

It can be shown that the transpose of the Jacobian maps the forces applied to the end–effector into joint torques [19]. The relationship is expressed below:

$$\tau = J^T f \tag{B.2}$$

where f is a 6×1 vector of forces and torques in Cartesian space and τ is a $n \times 1$ vector of torques, one for each joint of the robot arm.

As mentioned in the Notation section of this document, the Jacobian has several frames related to it. For example, ${}^kJ_{i,j}$ is the Jacobian relating the joint velocities of the robot arm to the Cartesian velocity of frame j with respect to frame i, and expressed in the coordinates of frame k. Frames i, j, and k are referred to as the velocity, reference, and coordinate frames, respectively.

B.3 Velocity and Coordinate Frame Transformations

B.3.1 Velocity Frames

The velocity frame of the Jacobian can be changed through the following transformation:

$$^{k}J_{i,l} = \begin{bmatrix} I & -^{k}\tilde{p}_{j,l} \\ 0 & I \end{bmatrix} {}^{k}J_{i,j}$$

Note that in the above equation, the coordinate frames i and k are arbitrary and not affected.

A special case of this transformation is analogous to the kinematic concept of a tool transform. Given the Jacobian which maps to the velocity of frame 9 of the robot arm ($J_{3,9}$ for the PUMA), and the position vector $^{9}p_{9,E}$ (obtained from the tool transform $^{9}_{E}T$), the Jacobian which maps to the end-effector velocity ($J_{3,E}$) can be found.

B.3.2 Coordinate Frames

In addition to velocity frame modifications, coordinate frame transformations can also be accomplished, via the following expression:

$$^{m}J_{i,j} = \begin{bmatrix} ^{m}_{k}R & 0 \\ 0 & ^{m}_{k}R \end{bmatrix} {}^{k}J_{i,j}$$

Combining both of these transformations, we have the following equation:

$$^{m}J_{i,l} = \begin{bmatrix} ^{m}_{k}R & 0 \\ 0 & ^{m}_{k}R \end{bmatrix} \begin{bmatrix} I & -^{k}\tilde{p}_{j,l} \\ 0 & I \end{bmatrix} {}^{k}J_{i,j}$$

B.3.3 Use of Transformations

Finding the Jacobian and its inverse with respect to any arbitrary coordinate frame can be computationally expensive. However, it is possible to take advantage of coordinate frame transformations to find the Jacobian matrix that has the the simplest form [22]. For PUMA arms, the Jacobian matrix is simplest when expressed in frame 6. This matrix is displayed below:

$$
{}^6J_{3,9} = \begin{bmatrix}
-(d_5+d_6)C_{56} & d_7 + a_5S_6 & d_7 & 0 & 0 & 0 \\
(d_5+d_6)S_{56} & a_6 + a_5C_6 & a_6 & 0 & 0 & 0 \\
a_5C_5 + a_6C_{56} + d_7S_{56} & 0 & 0 & 0 & 0 & 0 \\
-S_{56} & 0 & 0 & 0 & -S_7 & C_7S_8 \\
-C_{56} & 0 & 0 & -1 & 0 & -C_8 \\
0 & 1 & 1 & 0 & C_7 & S_7S_8
\end{bmatrix} \tag{B.3}
$$

Note also that the matrix in (B.3) is in lower block triangular form. This is due to the geometry of spherical wrist arms; i.e., the fact that the origins of the last three frames coincide. The following compact notation will be used to denote the matrix ${}^6J_{3,9}$:

$$
{}^6J_{3,9} = \begin{bmatrix} B & 0 \\ D & E \end{bmatrix}
$$

where B, D, and E are 3×3 submatrices of the PUMA Jacobian. Fijany and Bejczy [22] have shown that this structure can be utilized to obtain efficient solutions to equation (B.1). Instead of computing the Jacobian explicitly and then multiplying the Jacobian by a vector, the result of multiplying the Jacobian by a vector is computed directly. This strategy results in a significant computational savings.

B.4 Forward Jacobian

The equation relating the joint velocities $\dot{q}$ to the Cartesian velocity of frame E, with respect to frame k is:

$$^{k}\dot{x}_{E} = {}^{k}J_{3,E}\dot{q} \tag{B.4}$$

Separating the linear and angular components, we can rewrite this equation in the following matrix form:

$$\begin{bmatrix} ^{k}\dot{x}_{L} \\ ^{k}\omega \end{bmatrix} = \begin{bmatrix} ^{k}_{6}R & 0 \\ 0 & ^{k}_{6}R \end{bmatrix} \begin{bmatrix} I & -^{6}\tilde{p}_{9,E} \\ 0 & I \end{bmatrix} \begin{bmatrix} B & 0 \\ D & E \end{bmatrix} \begin{bmatrix} \dot{q}_{1} \\ \dot{q}_{2} \end{bmatrix} \tag{B.5}$$

The vector $\dot{q}_1$ is a 3×1 vector consisting of the velocities of the first three joints of the PUMA. Likewise, $\dot{q}_2$ contains the velocities of the last three joints of the PUMA. Multiplying the matrices together, we obtain the following equations:

$$\begin{aligned} ^{k}\dot{x}_{L} &= {}^{k}_{6}RB\dot{q}_{1} - {}^{k}_{6}R\,^{6}\tilde{p}_{9,E}D\dot{q}_{1} - {}^{k}_{6}R\,^{6}\tilde{p}_{9,E}E\dot{q}_{2} \\ ^{k}\omega &= {}^{k}_{6}RD\dot{q}_{1} + {}^{k}_{6}RE\dot{q}_{2} \end{aligned}$$

To simplify the equations (and to reduce the number of operations needed to compute the solution), we order the equations as follows:

$$\begin{aligned} ^{k}\omega &= {}^{k}_{6}R(D\dot{q}_{1} + E\dot{q}_{2}) \\ ^{k}\dot{x}_{L} &= {}^{k}_{6}RB\dot{q}_{1} - {}^{k}_{6}R\,^{6}\tilde{p}_{9,E}D\dot{q}_{1} - {}^{k}_{6}R\,^{6}\tilde{p}_{9,E}E\dot{q}_{2} \end{aligned} \tag{B.6}$$

B.4.1 Coordinate Frame $k \leq 6$:

When $k \leq 6$, equations (B.6) simplify easily:

$$\begin{aligned} ^{k}\omega &= {}^{k}_{6}R(D\dot{q}_{1} + E\dot{q}_{2}) \\ ^{k}\dot{x}_{L} &= {}^{k}_{6}R(B\dot{q}_{1} - {}^{6}\tilde{p}_{9,E}(D\dot{q}_{1} + E\dot{q}_{2})) \end{aligned}$$

And in their final form, the equations for $k \leq 6$ become:

$$\begin{aligned} s &= D\dot{q}_1 + E\dot{q}_2 \\ {}^{k}\omega &= {}^{k}_{6}Rs \\ {}^{k}\dot{x}_L &= {}^{k}_{6}R(B\dot{q}_1 - {}^{6}\tilde{p}_{9,E}\, s) \end{aligned} \quad \text{(B.7)}$$

B.4.2 Coordinate Frame $k > 6$:

When $k > 6$, a different form of the equations can be found which require less computation than the previous set. By using the property of cross product matrices (see Notation and Conventions) equations (B.6) become:

$$\begin{aligned} {}^{k}\omega &= {}^{k}_{6}R(D\dot{q}_1 + E\dot{q}_2) \\ {}^{k}\dot{x}_L &= {}^{k}_{6}RB\dot{q}_1 - {}^{k}\tilde{p}_{9,E}\,{}^{k}_{6}RD\dot{q}_1 - {}^{k}\tilde{p}_{9,E}\,{}^{k}_{6}RE\dot{q}_2 \end{aligned}$$

Regrouping the terms in the equation for ${}^{k}\dot{x}_L$, we obtain:

$$\begin{aligned} {}^{k}\omega &= {}^{k}_{6}R(D\dot{q}_1 + E\dot{q}_2) \\ {}^{k}\dot{x}_L &= {}^{k}_{6}RB\dot{q}_1 - {}^{k}\tilde{p}_{9,E}\,{}^{k}_{6}R(D\dot{q}_1 + E\dot{q}_2) \end{aligned}$$

The final form of the equations, for the case when $k > 6$ is:

$$\begin{aligned} {}^{k}\omega &= {}^{k}_{6}R(D\dot{q}_1 + E\dot{q}_2) \\ {}^{k}\dot{x}_L &= {}^{k}_{6}RB\dot{q}_1 - {}^{k}\tilde{p}_{9,E}\,{}^{k}\omega \end{aligned} \quad \text{(B.8)}$$

B.4.3 Without Tool Transform

When there is no tool transform, equations (B.6) and (B.8) simplify still further, into a common set of equations:

$$
\begin{aligned}
{}^{k}\omega &= {}^{k}_{6}R(D\dot{q}_1 + E\dot{q}_2) \\
{}^{k}\dot{x}_L &= {}^{k}_{6}RB\dot{q}_1
\end{aligned}
\tag{B.9}
$$

B.5 Forward Jacobian Transpose

The equation relating the Cartesian forces felt at frame E (and expressed in frame k) ${}^{k}f_E$, to the joint torques τ is:

$$\tau = {}^{k}J^{T}_{3,E}\,{}^{k}f_E \tag{B.10}$$

Separating the linear and angular portions of this equation, and rewriting in matrix form results in:

$$
\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} B^T & D^T \\ 0 & E^T \end{bmatrix} \begin{bmatrix} I & 0 \\ {}^{6}\tilde{p}_{9,E} & I \end{bmatrix} \begin{bmatrix} {}^{6}_{k}R & 0 \\ 0 & {}^{6}_{k}R \end{bmatrix} \begin{bmatrix} {}^{k}f_L \\ {}^{k}f_A \end{bmatrix} \tag{B.11}
$$

In the above equation. ${}^{k}f_L$ is a 3×1 vector containing the linear components of the force. ${}^{k}f_A$ is also a 3×1 vectors, and contains the components of the torque.

Multiplying the matrices together, we obtain the following equations:

$$
\begin{aligned}
\tau_1 &= B^T\,{}^{6}_{k}R\,{}^{k}f_L + D^T\,{}^{6}\tilde{p}_{9,E}\,{}^{6}_{k}R\,{}^{k}f_L + D^T\,{}^{6}_{k}R\,{}^{k}f_A \\
\tau_2 &= E^T\,{}^{6}\tilde{p}_{9,E}\,{}^{6}_{k}R\,{}^{k}f_L + E^T\,{}^{6}_{k}R\,{}^{k}f_A
\end{aligned}
$$

Regrouping terms, the equations become:

$$\begin{aligned}\tau_1 &= B^T {}^6_kR\, {}^kf_L + D^T({}^6\tilde{p}_{9,E}\, {}^6_kR\, {}^kf_L + {}^6_kR\, {}^kf_A) \\ \tau_2 &= E^T({}^6\tilde{p}_{9,E}\, {}^6_kR\, {}^kf_L + {}^6_kR\, {}^kf_A) \qquad \text{(B.12)}\end{aligned}$$

B.5.1 Coordinate Frame $k \leq 6$:

When $k \leq 6$, equations (B.12) simplify quickly to:

$$\begin{aligned}s &= {}^6\tilde{p}_{9,E}\, {}^6_kR\, {}^kf_L + {}^6_kR\, {}^kf_A \\ \tau_1 &= B^T {}^6_kR\, {}^kf_L + D^T s \\ \tau_2 &= E^T s \qquad \text{(B.13)}\end{aligned}$$

B.5.2 Coordinate Frame $k > 6$:

As with the Jacobian equations, when $k > 6$, a different form of the equations can be found which requires less computation. By using the property of cross product matrices, equations (B.12) become:

$$\begin{aligned}\tau_1 &= B^T {}^6_kR\, {}^kf_L + D^T {}^6_kR\, {}^k\tilde{p}_{9,E}\, {}^kf_L + D^T {}^6_kR\, {}^kf_A \\ \tau_2 &= E^T {}^6_kR\, {}^k\tilde{p}_{9,E}\, {}^kf_L + E^T {}^6_kR\, {}^kf_A\end{aligned}$$

Regrouping terms results in the following:

$$\begin{aligned}\tau_1 &= B^T {}^6_kR\, {}^kf_L + D^T {}^6_kR({}^k\tilde{p}_{9,E}\, {}^kf_L + {}^kf_A) \\ \tau_2 &= E^T {}^6_kR({}^k\tilde{p}_{9,E}\, {}^kf_L + {}^kf_A)\end{aligned}$$

So the final form of the equations for the case when $k > 6$ is:

$$\begin{aligned} s &= {}^{6}_{k}R({}^{k}\tilde{p}_{9,E}\,{}^{k}f_{L} + {}^{k}f_{A}) \\ \tau_1 &= B^{T}\,{}^{6}_{k}R\,{}^{k}f_{L} + D^{T}s \\ \tau_2 &= E^{T}s \end{aligned} \qquad \text{(B.14)}$$

B.5.3 Without Tool Transform

When there is no tool transform, equations (B.13) and (B.14) simplify into a common set of equations:

$$\begin{aligned} s &= {}^{6}_{k}R\,{}^{k}f_{A} \\ \tau_1 &= B^{T}\,{}^{6}_{k}R\,{}^{k}f_{L} + D^{T}s \\ \tau_2 &= E^{T}s \end{aligned} \qquad \text{(B.15)}$$

APPENDIX C
Mass Matrix and Control Gains For the PUMA Joint Controller

This chapter presents some additional information on the joint control layer of the force control code described in Chapter 2. In addition, the PID and PD feedback gains for the controller are presented. Finally, the simplified mass matrix of the PUMA which is used in the joint controller is presented.

C.1 Notation and Conventions

The notation in this chapter is consistent with that outlined in previous chapters. The reader is directed to Appendix A for a summary of the conventions.

C.2 Background

The joint controller which was described in Chapter 2 has several modes of operation. Among these modes are the PID and PD modes, which require joint velocity information.

Because of the design of the Unimate controller units, only joint position information can be obtained. Velocity information is not readily available. In order to implement the PID or PD modes of the joint controller, it is necessary to determine the joint velocities in some manner.

The method currently employed by the joint control layer is to use a simple-differencing approximation to obtain the velocity:

$$\dot{\theta}[k] \approx \frac{\theta[k] - \theta[k-1]}{T_s} \tag{C.1}$$

where T_s is the sampling period, in seconds.

Because this operation is a discrete–time differentiation process, the resulting velocity signal is very noisy. The velocity signal is then put into a discrete-time, low-pass filter in order to eliminate some of the noise. To minimize lag in the velocity signal, only a first-order filter is used. The filter equation which is used is:

$$\dot{\theta}_{filt}[k] = \frac{\omega_c T_s \, \dot{\theta}[k] + \dot{\theta}_{filt}[k-1]}{1.0 + \omega_c T_s} \tag{C.2}$$

where θ_{filt} is the filtered joint velocity signal, and ω_c is the desired (continuous-time) cutoff frequency in units of radians/second. The filtered velocity signal is used in the derivative term of the PID and PD feedback control laws.

C.3 PID Control Gains

Table C.1 lists the control gains and the desired filter cutoff frequencies for the PID control mode. These gains are used any time that the joint controller is in PID mode. During the force control experiments, prior to executing the force control algorithms, the robot arm was positioned using PID mode.

Arm Joint	Proportional Gain	Integral Gain	Derivative Gain	Filter Cutoff Frequency, ω_c (rad/sec)
4	210.9375	843.75	18.9844	60.0
5	240.0	1024.0	20.25	64.0
6	240.0	1024.0	20.25	64.0
7	540.0	3456.0	30.375	96.0
8	540.0	3456.0	30.375	96.0
9	540.0	3456.0	30.375	96.0

Table C.1: PID Control Gains and Filter Cutoff Frequency (Sampling Rate = 4.5 milliseconds)

C.4 PD Control Gains

Table C.2 lists the control gains and the desired filter cutoff frequencies for the PD control mode. PD mode was used exclusively during the execution of the

PAC control algorithm.

Arm Joint	Proportional Gain	Derivative Gain	Filter Cutoff Frequency (rad/sec)
4	108.0	16.0	54.0
5	108.0	16.0	54.0
6	108.0	16.0	54.0
7	432.0	32.0	108.0
8	432.0	32.0	108.0
9	432.0	32.0	108.0

Table C.2: PD Control Gains and Filter Cutoff Frequency (Sampling Rate = 4.5 milliseconds)

C.5 Approximation of PUMA Mass Matrix

The mass matrix of the PUMA is configuration dependent. Because of this, the performance of the joint controller will vary depending upon the configuration of the robot. To reduce the configuration dependence of joint controller when in PID or PD modes, an estimate of the mass matrix is computed and used in the feedback control law. This matrix is calculated every sampling period that the controller executes (4.5 ms).

The estimate of the mass matrix is shown below. It is greatly simplified; only the diagonal (dominant) terms of the matrix are calculated. The actual mass matrix of the PUMA is a symmetric non-diagonal matrix. The complete equations for the mass matrix can be found in [23, 26].

The equations and numerical values were obtained from [23]. The constants were modified to take into account the mass of the gripper and force sensor attached to the arm.

$$\widehat{M}(\theta) = \begin{bmatrix} a_{44} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{55} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{66} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{77} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{88} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{99} \end{bmatrix} \tag{C.3}$$

where:

$$\begin{aligned}
a_{44} &= 1.58 + 1.441\, C_5 C_5 + 0.6474\, S_{56} S_{56} - 0.04847\, S_{56} C_{56} + \\
&\quad -0.05977\, (S_8 S_8\, (S_{56} S_{56}\, (1.0 + C_7 C_7) - 1.0) - 2.0\, S_{56} C_{56} C_7 S_8 C_8) + \\
&\quad 1.3322\, C_5 S_{56} - 0.04898\, C_5 C_{56} + 0.1819\, (S_{56} S_{56} C_8 + S_{56} C_{56} C_7 S_8) + \\
&\quad 0.1814\, C_5\, (S_{56} C_8 + C_{56} C_7 S_8) + 0.06262\, S_7 S_8 \\
a_{55} &= 7.4461 + 1.3322\, S_6 - 0.05977\, S_7 S_7 S_8 S_8 - 0.04898\, C_6 + \\
&\quad 0.1819\, C_8 + 0.1814\, (S_6 C_8 + C_6 C_7 S_8) \\
a_{66} &= 1.5151 - 0.05977\, S_7 S_7 S_8 S_8 + 0.1819\, C_8 \\
a_{77} &= 0.2028 - 0.0598\, S_8 S_8 \\
a_{88} &= 0.2404 \\
a_{99} &= 0.1942
\end{aligned}$$