RICE UNIVERSITY

**Improved Haptic Fidelity
Via Reduced Sampling Period with an
FPGA-Based Real-Time Hardware Platform**

by

Emilie Kopp

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

MARCIA K. O'MALLEY, CHAIR

Assistant Professor

Mechanical Engineering and Materials Science

FATHI GHORBEL

Professor

Mechanical Engineering and Materials Science

LYDIA KAVRAKI

Noah Harding Professor

Computer Science

HOUSTON, TEXAS

JANUARY 2007

UMI Number: 1441831

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1441831

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Improved Haptic Fidelity
Via Reduced Sampling Period with an
FPGA-Based Real-Time Hardware Platform
by
Emilie Kopp

The challenge of displaying stiff virtual walls in a haptic interaction is the inspiration for this document. The fidelity of their implementation necessitates high virtual stiffness integrated with passive device behavior. An energy-based approach to system passivity for a unilateral, virtual spring is derived. The work addresses the issue of sampling rate; derivations reveal that increasing control loop rates can increase achievable displayed stiffness.

The objective is to directly minimize the computational period of the haptic rendering loop for a unilateral virtual spring by employing a hardware platform that utilizes a Field Programmable Gate Array (FPGA) device to handle high-speed data acquisition and low-level control executions directly on hardware. A dedicated real time (RT) execution target coupled to the FPGA is proposed to manage higher-level, deterministic operations necessary for virtual wall implementation in software.

The proposed FPGA/RT platform is interfaced with the PHANToM Premium 1.0 commercial haptic manipulator and achieves control loop rates 20 times faster than those executed by its default hardware platform. Experimental findings validate the improved performance of virtual spring rendering. The FPGA/RT platform successfully displays virtual stiffness that far exceeds the passive range of the PHANToM's commercial hardware platform.

Table of Contents

1	Introduction	1
1.1	Haptic Feedback	2
1.2	Haptic Implementation	3
1.3	Impedance-Based Interactions	5
1.4	The Virtual Wall	6
1.5	Maintaining Passivity.....	7
1.5.1	Compensation Methods	8
1.5.2	Direct Methods.....	10
1.6	Objective	11
2	Virtual Wall Passivity	14
2.1	System Model	14
2.2	Passivity Proof	19
2.2.1	Energy-Approach Derivation	19
2.2.2	Condition Interpretation	21
3	Performance Improvement	27
3.1	Minimizing Sampling Period	28
3.1.1	Decision Making in Software vs. Hardware	29
3.1.2	Proposed Platform Solution	31
4	Implementation	33
4.1	Haptic Device: The PHANToM Premium	33
4.1.1	Impedance-Based Implementation	35
4.1.2	Forward Kinematics	35
4.1.3	Inverse Kinematics	39
4.1.4	Jacobian	40
4.1.5	Dynamics	41
4.1.6	Electronic Properties	43
4.2	FPGA-Based Hardware Platform	44
5	Experiment	48
5.1	Setup	48
5.2	Results	49
5.3	Observations	55
6	Conclusions	65
7	References	66

Appendix A: Forward Kinematics Verification	69
A.1: Forward Kinematics Test M-file	
Appendix B: Dynamic Torque Assumption Validation	72
B.1: Negligible End Effector Force Calculation M-file	
B.2: Negligible End Effector Force Command Window	
B.3: Inverse Jacobian Transpose M-file	
Appendix C: PHANToM Communication Wire Map	78
PHANToM Communication Cable Wire Map	
Appendix D: Platform Hardware	80
D.1: NI PXI-7831R Specifications	
Appendix E: Platform Software	82
E.1 LabVIEW FPGA Virtual Instrumentation	
E.2 LabVIEW RealTime Virtual Instrumentation	
E.3 LabVIEW RealTime Front Panel	

List of Figures

1.1	Human-Computer Interaction using Haptic Information	2
1.2	Impedance-Based Haptic Interaction	3
2.1	Virtual Wall Implementation Model [9]	14
2.2	Spring Force Model when Compressed	16
2.3	Spring Force Model when Released	17
2.4	Total Energy Generation when Contacting a Virtual Wall	18
2.5	Virtual Wall Threshold Switching	18
2.6	Simple Spring-Mass System	20
2.6	Moving Out of the Virtual Wall	23
2.7	Moving Into Virtual Wall	24
3.1	Typical Communication Path when Controlled in Software	29
3.2	Optimized Communication Path when Controlled in Hardware	30
4.1	PHANToM Premium 1.0 [21]	33
4.2	Impedance-Based Control Block Diagram	35
4.3	Rotational Axes of PHANToM	36
4.4	Decoupling the PHANToM Joint Angles	37
4.5	Serial Axes of Rotation	37
4.6	Assigned Transformation Frames	38
4.7	PHANToM Top View [23]	39
4.8	PHANToM Side View [23]	40
4.9	PHANToM Commercial Package Setup	43
4.10	Custom Hardware Platform Overview (adapted from [20])	46
5.1	Drop Test Setup	48
5.2	Spring Mass Drop Test Response with Virtual Stiffness of 1 N/m	
a.	Loop Rate of 1kHz	50
b.	Loop Rate of 20 kHz	50
5.3	Spring Mass Drop Test Response with Virtual Stiffness of 5 N/m	
a.	Loop Rate of 1kHz	51
b.	Loop Rate of 20 kHz	51
5.4	Spring Mass Drop Test Response with Virtual Stiffness of 10 N/m	
a.	Loop Rate of 1kHz	52
b.	Loop Rate of 20 kHz	52
5.5	Spring Mass Drop Test Response with Virtual Stiffness of 25 N/m	
a.	Loop Rate of 1kHz	53
b.	Loop Rate of 20 kHz	53
5.6	Spring Mass Drop Test Response with Virtual Stiffness of 50 N/m	
a.	Loop Rate of 1kHz	54
b.	Loop Rate of 20 kHz	54
5.7	Spring Mass Response with Virtual Stiffness of 100 N/m at 20 kHz	56

5.7	Spring Mass Response with Virtual Stiffness of 200 N/m at 20 kHz	56
5.7	Spring Mass Response with Virtual Stiffness of 300 N/m at 20 kHz	56
5.7	Spring Mass Response with Virtual Stiffness of 600 N/m at 20 kHz	57
5.7	Spring Mass Response with Virtual Stiffness of 100 N/m at 20 kHz	57

A.1 Forward Kinematics Verification Simulation Results

1 Introduction

Derived from the Greek word *haptesthai* (meaning to touch), haptics appropriately refers to stimulation of the human central nervous system in the form of touch sensations. This physical contact provides the ability for humans to perceive and manipulate the world around them.

In the past decade, the realm of physical perception has crossed over into the virtual world through implementation of robotic haptic devices. These machines can serve as a direct interface between human and computer, transmitting force feedback to convey physical traits of a virtual environment.

Prior to the recent interest in haptic interfaces, humans typically interacted with computers through the click of a mouse or the press of a button. Any processed response from the computer would be in the form of visual display (a blink of an icon) or auditory cues (beeps, dings, etc.), completing a unidirectional, disjointed loop of interaction. Moreover, representation of a virtual environment was limited to the quality of visual rendering, relying on the eye's sensitivity to changes in pixilation, for instance, in order to detect changes in the displayed information.

Haptic feedback, as an alternative form of communication between human and computer, is unique in that a bi-directional exchange of information in the form of mechanical energy occurs at a single continuous junction.

Figure 1.1: Human-Computer Interaction using Haptic Information

In this new interaction loop, the computer communicates directly with the human operator, and vice versa, through the haptic interface, utilizing physical stimuli to expand and refine the representation of a virtual environment.

This leap in the evolution of virtual reality has led to an enormous amount of interest from researchers in a wide range of disciplines, all of which seek a better understanding and utilization of haptic techniques.

1.1 Haptic Feedback

There are two types of haptic feedback employed when displaying features of a virtual object to a human operator. One mode, tactile feedback, stimulates cutaneous receptors embedded in the skin, conveying virtual surface features like temperature, pressure, roughness and vibration. The other mode, kinesthetic feedback, is detected by proprioceptive sensors embedded in muscles, joints and tendons, resulting in perception of force features like weight, momentum and stiffness. This latter mode is the most common form of feedback implemented in haptic simulations due to the availability and quality of force-reflecting robotic devices, and is also the focus of this document.

1.2 Haptic Implementation

Any haptic interaction must employ an interface through which the exchange of mechanical energy between human and computer can flow. A haptic manipulator can serve as this physical connection, exerting precisely controlled dynamic responses to the user's grasp at the end-effector, providing the sensation of contact and manipulation of simulated virtual objects. These robotic devices can vary in the number of degrees of freedom and associated applications.

The causality of interaction is often what sets the variety of haptic devices apart. An impedance haptic device will detect a user's motion (position, velocity, acceleration) and output an appropriate force vector to simulate interactions with virtual objects.

Figure 1.2: Impedance-Based Haptic Interaction

Effectively, the haptic interface acts to impede the user motion when in contact with a virtual surface. These devices require transparent constraints on motion, such as minimal intrinsic friction and back-drive inertia, so that the sensation of free space actually feels free [1].

In addition to providing the medium for force-position control, impedance devices are now being used to display an alternative form of force-reflection called event-based haptics. This architecture attempts to achieve stable, high-gain force reflection via

cancellation of a user's induced motion by applying playback of previously recorded high-frequency responses [2]. However, this technique of force-reflection is relatively unfamiliar in the haptics community and still remains in its early stages of implementation.

An admittance haptic device can be considered the exact opposite of the impedance device in that it detects a user's force vector and outputs an appropriate position. Although these devices typically possess the ability to display high forces in large workspaces, impedance devices are generally simpler to design and cheaper to produce, making them more common in the haptics community [1].

Any effective force-reflecting device must meet the demanding, and sometimes conflicting, standards of a high-performance haptic interface. Although there are no quantitative specifications for these manipulators, it is evident from literature that haptic researchers seek a device with the following features [3]:

- Low apparent mass/inertia
- Low friction
- High structural stiffness
- Backdrivability
- Zero or very little backlash
- High force bandwidth
- High force dynamic range
- Absence of mechanical singularities
- Accessibility to the operator
- Compactness

- Even/symmetric “feel” through the workspace

The specific task of each haptic device governs which, and to what extent, these demands will be met.

1.3 Impedance-Based Interactions

Impedance-based interactions are the framework of haptic feedback in this document. Obeying this causality, rendering a virtual environment entails computing appropriate interaction forces between the human operator and the virtual objects populating the environment according to the current position state vector. These impedance forces can be calculated using dynamic equations that possess the desired mechanical features of the real-world object that the haptic interface aims to simulate. Upon collision with a virtual surface, mathematical representations of mass, stiffness, damping, etc. are conveyed to the user through the force-reflecting interface.

A universal facet of haptic rendering is the virtual wall. This fundamental building block is typically modeled as a simple, unilateral spring. Contact with a stiff virtual wall dictates the immediate and reversible switch between little or no impedance (free motion) to the sensation of infinite stiffness (rigid surface). The reaction force can be calculated with Hooke’s Law: simply a linear product of virtual spring stiffness and the user’s displacement of the virtual surface.

Quantitative measures regarding the quality of virtual wall implementation have yet to be explicitly defined in the haptics research field. They become a question of what makes a virtual wall “feel” good and can vary between each operator since each individual can possess unique levels of perception [4].

Psychophysical experiments have been conducted to characterize the perceptual qualities of virtual wall performance. The work of Rosenberg and Adelstien [5] evaluates a series of haptic wall simulations, rating the overall performance based on the subjects' cognition of initial crispness of contact, hardness of surface rigidity, and cleanness of final release.

Still, little can be classified from a dynamics and controls approach to these qualities. What is known is that the quality of performance is often limited by the dynamic range of impedances a haptic simulation is capable of displaying [6]. Fundamentally, the fidelity of a rigid virtual wall is dependent upon the virtual stiffness perceived by the human operator.

Yet despite the apparent simplicity of implementation, the rendering of a fidelic, stiff virtual surface has become a ubiquitous challenge in the field of haptics.

1.4 The Virtual Wall

Intuitive reasoning behind why virtual walls often evade convincing renderings is illuminated by Colgate et al. [6]. Because of the inevitably discrete nature of implementation, a virtual spring always acts to generate or “leak” energy into the simulated system, thereby failing to capture the inherently passive (dissipative) nature of a physical spring of identical stiffness.

Previous research chronicling these energy leaks begins with their initial recognition by Kazerooni [7] as a consequence of sampled-data implementation. Further insight to this time-variance is provided by Gillespie and Cutosky [8], detailing the ramifications of the zero-order hold (ZOH) and asynchrony of continuous and discrete traversal through the virtual wall threshold.

The catalog of virtual energy generation is further refined to include the effects of position quantization. The work of Abbott and Okamura [9] provides explicit evidence that the quantized position signal necessary for impedance-based rendering can contribute to the non-passive nature of a virtual spring. Diolatti et al. [10] affirm this outcome but maintain that the energy generation is a result of a position signal that is both quantized and discretized, and not just the former.

In actuality, when rendering modest stiffness, the energy leaks are sufficiently dissipated by intrinsic device friction and thus remain transparent to the human operator. But rigid virtual walls call for especially high stiffness, and as virtual stiffness is increased, the energy generated by the virtual system exceeds the energy dissipated.

1.5 Maintaining Passivity

When dissipative elements can no longer dominate the virtual energy seepage small vibrations or rumbles emanate at the haptic device interface, active behaviors that even in the slightest magnitude can be immediately detected by the operator; the proprioceptive sensors found in the hand are extremely receptive to small amplitude mechanical vibrations in the 100 Hz- 1 kHz range [11]. An unfortunate consequence of this keen sensitivity to energy leaks is that any illusion of reality possessed by the user is instantly spoiled. But perhaps of even greater importance: because of the physical exchange of energy between the operator and device, sustained and/or growing oscillations induced by non-passive behaviors can damage hardware or even worse, inflict physical harm to the human.

Thus, the concept of passivity has become a widely-accepted property of robustness in virtual wall implementation. It provides a criterion that, through derivation

of its proof, defines an explicit upper bound on achievable virtual stiffness of a haptic interaction as function of sample period, position sensor resolution, viscous damping and Coulomb friction [11, 9].

Creating a virtual wall that behaves passively has subsequently spawned extensive research dedicated to alleviating its limitation enforced on achievable stiffness. Often times, the challenge can become a compromise or submission of one or more of the high-performance haptic interaction standards recommended by Ellis et. al [3].

1.5.1 Compensation Methods

For example, a conventional remedy for energy leaks that can degrade virtual stiffness perception is to simply compensate with more physical dissipation. Miller, Colgate and Freeman [12] promote this strategy, defining a function to determine the amount of inherent damping a haptic device should possess to sufficiently dissipate unwanted energy generation. However, excessive damping can compromise overall transparency, making free space no longer feel “free”.

The work of Colgate and Brown [11] foresees this caveat and suggests countering it with negative virtual damping. They admit, however, that this novel solution relies on exact cancellation of device friction with negative virtual damping, a precision that is difficult to achieve in practice and even then, the system is borderline passive.

Shen and Goldfarb [13] instead propose to leverage open-loop stiffness of pneumatic actuators to assist displayed stiffness of a passive haptic simulation. They derive a passivity analysis similar to that pioneered in [11] that includes inherent device stiffness to expand the region of passivity for a stiff simulated surface. Their design employs a pneumatically actuated haptic interface and while its performance is indeed

exceptional in displaying a rigid surface ($K=10,000$ N/m), the complexity of work space is limited to one degree of freedom; utilizing a similar tactic for a multi-dimensional device would require several linear actuators and would most likely tax the dexterity and available workspace.

An alternative to the modification of device hardware to tame the flow of system energy is the utilization of simulation software to avoid detection of leaks by the human operator. Colgate et al. [6] dispose of unwanted energy through the process of “virtual coupling,” a scheme that modifies the simulated environment by means of a filtering mechanism to ensure that the passivity of the system is withheld. Hannaford and Ryu [14] embrace a similar approach of preserving system passivity, deploying an online passivity measure/observer which appropriately prompts a passivity controller to adjust an adaptive and dissipative rendering element.

Other kindred methods of compensation that do not explicitly maintain passivity have also been developed. Gillespie and Cutosky [8] attempt to maintain virtual wall stability rather than passivity, a less conservative approach to avoiding virtual energy generation. They monitor the energy-hemorrhage effects of zero order hold and asynchronous switching of the spring-force surface model with a watchdog algorithm, periodically eliminating them as necessary with a deadbeat controller.

It is worth noting here that the notions of passivity and stability are both similar in their attempt to maintain bounded system energy but are fundamentally different in their foundations. A passive system does not necessarily guarantee a stable system, contrary to what its definition may suggest. While passivity analysis can produce a mathematical description of a controlled system’s power and energy, it based solely on input-output

properties; without any knowledge of the internal states, stability cannot be not justly inferred. However, it has been shown that humans unconsciously remain passive when interacting with passive systems [25]. Thus, passive control strategies constitute a sufficient condition for the stability of virtual systems coupled to any passive device excited by a bounded operator input energy. In regards to virtual wall implementation, it is proven in [9] that passivity is a necessary and sufficient condition for stability.

1.5.2 Direct Methods

Instead of compensating for energy generation with physical or virtual dissipation, perhaps a more direct method of extending the passivity region of a virtual spring is to choke the original source of leaks⁴, minimizing the disparity between the continuous spring model and its actual sampled-data system; in essence, decreasing the sampling period can minimize creation of unwanted energy. A quick fix is to simplify the rendering algorithm to alleviate the operating system of computationally heavy calculations [15], although this almost always sacrifices virtual scene complexity. Movash and Hayward [16] fuse this tactic with that of the previously mentioned passivity controller in [8], constructing local passive models from local field approximations instead of enforcing global passivity, resulting in an increase in servo rates of complex virtual environments.

Many haptic researchers have opted to streamline communication methods with a force-reflecting device by employing a real time (RT) operating system (OS), a type of event-driven OS that abides by a strict and unwavering hierarchy of task execution. Although this tactic has not yet been explicitly employed to upgrade the performance of

passive virtual walls, RT OS's have long since been a boon to sampled-data control systems in general. They are not typically revered as having exceptionally high throughput, but are utilized more for their ability to operate deterministically, guaranteeing completion of processes on a time deadline. Yet due to its strict governing of processing threads, an RT OS can often achieve algorithmic control rates faster than those of a typical OS. Of course this can prove an arduous task for those not fluent in software programming languages; most traditional real time systems are designed using ad hoc techniques and are customarily hand coded in assembly languages [17].

The attempt to directly increase the sampling rate of a haptic control loop in order to thwart the creation of unwanted virtual energy may not be limited to the remedies executed in the domain of software design. On the contrary, faster loop rates can be obtained by simply exploiting more capable processors to execute the haptic rendering loop. Likewise, one can boost computational speeds of a processor by eliminating any and all other operations from the payload of execution tasks, devoting all its resources towards one specific application. But to obtain either breed of processor, exceptionally powerful or highly specialized, would draw an equally exceptional/high price.

1.6 Objective

Some of the previous methods seek to maximize virtual stiffness using methods that compensate for virtual energy generation, while others seek to optimize rendering techniques in order to avoid energy generation altogether; the objective of this work is derived from the latter. In particular, it addresses the issue of virtual system passivity at the source of energy leaks.

Rather than altering the haptic rendering algorithm to increase computational speeds with scene-simplification or local surface modeling, as done in [15] and [16], the optimization technique presented in this document draws inspiration from the streamlined, deterministic capabilities of an RT OS.

However, this ambition builds beyond the enhancement of computation speeds with efficient software schemes; it engages the added computing power of dedicated computational hardware. Specifically, this work aims to minimize the control loop period by deconstructing the haptic device communication path, performing low-level read and write operations directly on hardware and handling the virtual rendering algorithm on a RT execution target.

A Field Programmable Gate Array (FPGA) chip is the computing hardware proposed to interface with a haptic manipulator, acting as a high-speed, custom data acquisition and processing device, thus bypassing the software protocols and application interfaces necessary for acquisition and control with typical operating systems (in particular, Windows OS). The signals processed on the FPGA are targeted by a separate processor running an RT OS where a standard, impedance-based rendering algorithm for virtual walls is executed.

To demonstrate the improvements in virtual wall display using this computational platform, a PHANToM Premium 1.0 force-reflecting manipulator (SensAble Technologies) is interfaced with an FPGA provided by National Instruments, demanding a rigorous investigation of the commercial device's electrical and mechanical architecture. The PHANToM is then operated from a RT execution target, also provided

by National Instruments. Both software and hardware resources are programmed using the graphical virtual instrumentation language of LabVIEW.

The custom FPGA-plus-RT OS (FPGA/RT) platform not only achieves control loop rates 20 times faster than the PHANToM's default hardware platform, but because the haptic rendering algorithm is executed on a dedicated, deterministic system, constant and continuous communication with the robotic device is now guaranteed.

A virtual wall test is conducted when operating the PHANToM on the default and custom platforms. Results show that the FPGA/RT platform is able to render passive interactions for virtual stiffness much greater than the displayable stiffness of the default haptic configuration.

Since the control loop period is significantly decreased with the dedicated processing hardware and fine-tuned software of the FPGA/RT platform, the passive range of virtual stiffness for the PHANToM is greatly expanded. Because the quality of a virtual wall is generally dependant upon the displayed stiffness and transparency of the simulation, the FPGA/RT hardware platform significantly improves the performance of the haptic device.

2 Virtual Wall Passivity

Due to the physical exchange of energy between a human user and a haptic device, it is beneficial to implement a haptic simulation that exhibits passive behavior. It is also desirable, however, to have the capability of displaying considerably high stiffness with the haptic device [6]. It is here that one must balance the conflicting aspirations of maximum achievable stiffness and maintaining a passive system. If one can extend the passivity region of operation into higher impedances, one can display stiffer, and hence, more fidelic surfaces in the haptic simulation [10].

2.1 System Model

To examine the maximum allowable stiffness for a passive virtual wall haptic simulation, first consider the following system model [9].

Figure 2.1: Virtual Wall Implementation Model [9]

Here, a 1 DOF haptic device is modeled as a mass acted upon by three separate forces:

- 1) The force inflicted by the human user, $f_h(t)$

- 2) The force applied by an actuator $f_a(t)$
- 3) The force due to device friction $f_f(t)$

These are the forces present when one implements any impedance-based virtual environment, specifically the virtual wall.

$$f_h(t) - f_a(t) - f_f(t) = m\ddot{x}(t) \quad (1)$$

The human exerts a positive force when pushing into the wall while the actuator and friction forces oppose this motion and are defined as positive when moving out of the wall.

The actuator force is modeled by Hooke's Law, a simple spring with a unilateral constraint,

$$f_a(t) = \begin{cases} Kx, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (2)$$

where K is the stiffness of the virtual wall and x is the measured position of penetration into the virtual wall. When $x \leq 0$, the user is not in contact with the wall. Thus there is no applied actuator force and the user should feel free space.

The chosen friction force model from [9] includes both Coulomb and viscous friction.

$$f_f(t) = \begin{cases} f_c \operatorname{sgn}(\dot{x}(t)) + b\dot{x}(t), & \dot{x} \neq 0 \\ \min(f_c, |f_e(t)|) \operatorname{sgn}(f_e(t)), & \dot{x} = 0 \end{cases} \quad (3)$$

Here, f_c and b are the positive, constant Coulomb and viscous friction parameters of the haptic device, respectively, and f_e is the net external force on the mass, i.e.

$$f_e = f_h(t) - f_a(t). \quad (4)$$

The closed loop model depicted in Fig. 2.1 is a sampled-data system, meaning the position signal necessary to calculate the actuator force modeled in Eq. (2) is quantized with a resolution of Δ , and the resulting f_a is implemented in discrete time with a constant sampling period, T . Therefore, the virtual wall represented by the expression in (2) is more appropriately modeled by

$$f_a(k) = \begin{cases} Kx_{quant}(k), & x_{quant}(k) > 0 \\ 0, & x_{quant}(k) \leq 0 \end{cases}, \quad (5)$$

where $x_{quant}(k)$ is the quantized measured position at sample k .

Because of the discrete nature of implementation, f_a is repeatedly held constant for the duration of each sampling period. This common event of holding a discrete state value across a sampling step is known as zero-order hold (or ZOH). The result is a stair-step-like, piece-wise-continuous actuator force that errs from the linearity of the real spring-force model.

Figure 2.2: Spring-Force Model When Compressed

Consider the implications of variance between sampled and continuous spring models: When the virtual spring is compressed, the deflection of the wall increases, warranting an increase in the calculated actuator force. This impeding force is calculated

at the beginning of the sample period and is held until an update of the next position sample. In this respect, when compressing a virtual spring, the force calculation slightly lags the increasing wall deflection, resulting in a virtual spring force marginally less than that of a physical spring following the same trajectory. More precisely, the work required to deflect the virtual spring is less than the necessary work for the identical deflection of a physical spring, the magnitude of which is equal to the difference in area underneath both curves in Fig. 2.2.

Equivalently, when releasing a virtual spring, the force calculation slightly lags the decreasing wall deflection, resulting in a virtual spring force always slightly greater than that of a physical spring:

Figure 2.3: Spring Force Model When Released

For both cases, assuming there is no initial difference in work invested in each spring system, the virtual spring introduces physical energy that is unaccounted for in its natural physics. As such, when the virtual wall is penetrated, the discrete simulation unavoidably “leaks” energy into the haptic implementation. The total magnitude of this virtual energy, represented by the grey area highlighted in Fig. 2.4, is equal to the work

slighted by the virtual spring during compression plus the energy procured during its release.

Figure 2.4: Total Energy Generation when Contacting a Virtual Wall

Kindred energy leaks occur due to the instant switch of actuation when making and breaking contact with a virtual wall.

Figure 2.5: Virtual Wall Threshold Switching

When initially making contact with the virtual wall, the haptic device should immediately display the physical reaction force of the spring. However, the actuator force

is not actually commanded to the device until the beginning of the time sample after contact was originally made. The virtual spring now stores energy without the requisite work having been done (observe the area between curves, *a, in Fig. 2.5).

Upon leaving the wall, the actuator force should immediately be turned off when the user crosses into free, unimpeded space. Yet in actuality, the commanded actuator force to the device continues to push the user away from the wall threshold for duration of the current sample period (observe the area bounded by *b). The asynchrony between on/off switching in continuous time and sampled time has a net-energy effect [8].

2.2 Passivity Proof

When the virtual wall model is applied to a haptic interaction, one can guarantee the system to be passive if and only if the virtual spring stiffness to be displayed does not exceed the minimum of two ratios: viscous damping to sampling rate and Coulomb friction to encoder resolution [11, 9].

$$K \leq \min\left(\frac{2b}{T}, \frac{2f_c}{\Delta}\right) \quad (6)$$

2.1.1. Energy Approach Derivation

To prove that this upper bound on virtual stiffness exhibits the domain of passive behavior of the device when implementing virtual wall, consider first a unilateral, ideal mass-spring system, pictured in Fig. 2.7.

Figure 2.76 Simple Spring-Mass System

The ideal nature of the mass-spring indicates that it is conservative, i.e. there is no energy lost in the system. On the other hand, the inanimate nature of the mass-spring demonstrates that it is inherently passive, i.e. there is no energy created by the system. As such, the ideal mass-spring exhibits flawless obedience to the governing laws of classical physics; the energy supplied to the system is always equal to the energy output by the system. Therefore, the total energy possessed by the system over any interval of time, $t \in [0, T]$, is equal to the sum of the potential energy contained by the spring and the kinetic energy exhibited by the mass.

$$W_{ideal} = \frac{1}{2} K (x_o^2 - x_T^2) + \frac{1}{2} m (\dot{x}_o^2 - \dot{x}_T^2) \quad (7)$$

Now imagine that an exact replica of this unilateral mass-spring system is to be virtually implemented using a haptic device in order to render a virtual wall. It is assumed that the two systems have the same mass, exhibit the same stiffness and follow the same trajectory over the same time interval (same initial $(x_o, \dot{x}_o)$ and final $(x_T, \dot{x}_T)$ states, where $x(t=0) = x_o$ and $x(t=T) = x_T$). The rendered physical energy displayed by the haptic device can be expressed as follows:

$$W_{virtual} = f_a (x_o - x_T) + \frac{1}{2} m (\dot{x}_o^2 - \dot{x}_T^2) - W_f, \quad (8)$$

where f_a is the constant, non-negative force imposed by the actuator to simulate the virtual spring of stiffness K , and W_f is a non-negative sum of unavoidable energy losses due to friction in the physical device.

For this virtual system to exhibit passive behavior, the total energy displayed at its operating point must never exceed the total energy of its ideal cousin.

$$W_{ideal} \geq W_{virtual} \quad (9)$$

Since identical system parameters are considered over identical trajectories, their kinetic energies are equal. The condition in (9) can then be expressed as the difference in each system's potential energy.

$$\frac{1}{2}K(x_0^2 - x_T^2) - (f_a(x_0 - x_T) - W_f) \geq 0 \quad (10)$$

If this condition is not satisfied, the virtual energy has exceeded that of the ideal case; energy has somehow leaked into the haptic simulation and can therefore result in non-passive behavior.

2.1.1 Condition Interpretation

Note in Eq. (8) that the energy loss due to inherent friction in the device reduces the total energy in the virtual system. Because this quantity tends to dissipate the generated energy and is thus beneficial to maintaining passive behavior of the virtual mass-spring, the conditions that minimize the effects of W_f should be examined.

The total friction loss in the device is a result of both Coulomb and viscous friction.

$$W_f = W_{f_c} + W_{f_v} \quad (11)$$

It is proven in [9] that if a monotonic trajectory with no stops in finite time is assumed, the friction losses are minimized. A result of this assumption is that the effects

of Coulomb friction, W_{f_c} , can be modeled as a constant retarding force. This eliminates the nonlinear switching nature of Coulomb friction. This quantity can then be expressed as:

$$W_{f_c} = f_c(x_o - x_T). \quad (12)$$

The lower bound on viscous friction losses is then considered.

$$W_{f_v} = \int_0^T b\dot{x}(t)dt \quad (13)$$

One can utilize the Cauchy-Swartz inequality [18] to find the lower bound of this quantity.

$$\left(\int_0^T \dot{x}(t)dt \right)^2 \leq T \int_0^T \dot{x}^2(t)dt \quad (14)$$

Thus,

$$W_{f_v, \min} = \left[b \int_0^T \dot{x}(t)dt \right]_{\min} = b \frac{1}{T} \left(\int_0^T \dot{x}(t)dt \right)^2 = \frac{b}{T} (x_T - x_o)^2. \quad (15)$$

The condition in (10) can now be more explicitly expressed as

$$\frac{1}{2} K(x_o^2 - x_T^2) - f_a(x_o - x_T) + f_c(x_o - x_T) + \frac{b}{T} (x_T - x_o)^2 \geq 0. \quad (16)$$

Now consider the work applied by the actuator force over the trajectory,

$$f_a = K(x_o - \delta), \quad (17)$$

where δ is an unknown, discrete position, $0 \leq \delta \leq \Delta$.

The maximum commanded actuator force occurs when $\delta = 0$. Therefore, let

$$f_a = Kx_o. \quad (18)$$

Now, (10) in its most conservative form, can be written as

$$\frac{1}{2}K(x_o^2 - x_T^2) - Kx_o(x_o - x_T) + f_c(x_o - x_T) + \frac{b}{T}(x_T - x_o)^2 \geq 0. \quad (19)$$

Expanding and simplifying this expression yields:

$$\left(\frac{b}{T} - \frac{K}{2}\right)(x_o - x_T)^2 + f_c(x_o - x_T) \geq 0. \quad (20)$$

Here, it is necessary to refine the definition of the trajectory.

Figure 2.7 Moving Out of the Wall

If one assumes the driving point of the device is released from within the virtual wall, where $x_o > x_T$ and $x_o > 0$, the following conclusions can be made regarding the expression in (20):

$$f_c(x_o - x_T) > 0 \quad (21)$$

and

$$(x_o - x_T)^2 > 0. \quad (22)$$

(Note that this trajectory type includes the case of a final position that is less deep within the wall, as well as the case of leaving the wall completely.)

Therefore, one can guarantee the rendered energy of a virtual spring will never exceed the potential energy of its ideal model when released from compression if

$$\frac{b}{T} - \frac{K}{2} \geq 0. \quad (23)$$

Rewriting this expression reveals an upper bound on virtual stiffness,

$$K \leq \frac{2b}{T}. \quad (24)$$

Regressing back to the definition of the trajectory, another possible scenario entails the spring being compressed further into the wall from its initial state, where $x_o < x_T$ and $x_o \geq 0$.

Figure 2.8: Moving Into the Wall

Note that in this circumstance, the sign of W_{fc} is negative since it opposes the motion now in the opposite direction:

$$W_{fc} = -f_c(x_o - x_T). \quad (25)$$

Similarly, if only the most extreme case of f_a is reconsidered, this quantity is maximized when $\delta = \Delta$. Hence,

$$f_a = K(x_o - \Delta). \quad (26)$$

It follows that, for the scenario of the virtual spring compressed from its initial state further into the wall, (10) is rewritten as:

$$\frac{1}{2}K(x_o^2 - x_r^2) - K(x_o - \Delta)(x_o - x_r) - f_c(x_o - x_r) + \frac{b}{T}(x_r - x_o)^2 \geq 0. \quad (27)$$

Expanding and simplifying this expression yields:

$$\left(\frac{b}{T} - \frac{K}{2}\right)(x_o - x_r)^2 + (K\Delta - f_c)(x_o - x_r) \geq 0. \quad (28)$$

It has been previously established that $\frac{b}{T} - \frac{K}{2} \geq 0$, so in order to maintain the condition in (28), the following additional condition must be met:

$$K\Delta - f_c \leq 0. \quad (29)$$

It is worthy to acknowledge that this resulting condition is actually over-conservative in its consideration. Although the virtual spring is compressed, it can only generate energy during the release phase. In other words, a virtual wall can only “push,” so it is not necessary for the virtual spring to be more dissipative than the ideal case during compression, as long as it remains significantly more dissipative during the release phase. A less conservative sufficient condition is provided by Abbot and Okamura [9]:

$$K \leq \frac{2f_c}{\Delta}. \quad (30)$$

From the time-domain energy analysis of a passive virtual wall, the condition in (6) is proven for an impedance-based haptic device. The explicit upper bound on virtual stiffness is a function of the two simple ratios in (24) and (30).

This condition also holds for trajectories not yet considered. For the case of crossing into the wall, where $x_o < 0$ and $x_r \geq 0$, the trajectory would acquire the same amount of potential energy as the previously mentioned case of moving further into the wall. Ergo, there is no gain in potential energy in the free space before the wall and once

inside the wall, the energy exchange of both trajectories is identical. Moreover, in addition to dissipating the same amount of energy once inside the wall, there is also the loss of energy due to friction when approaching the wall in free space. Intuitively, this trajectory would dissipate at least as much work as the amount of potential energy gained and would therefore never violate the conditions resulting from the trajectories previously considered.

Similarly, a trajectory that remains outside of the wall ($x_o < 0$ and $x_T < 0$) is entirely dissipative and does not necessitate an examination of the total energy possessed over the sampling period like that of the previously considered trajectories. It is obvious that this scenario will always remain passive: there is no applied actuator force, yet there is still loss of energy due to inherent device friction if there is any movement at all. This final trajectory case is trivial.

It can be reasonably assumed that the conditions expressed in (24) and (28) were derived from “worst-case” scenarios. Therefore, when considering the proposed system model, if the minimum of these two ratios is never exceeded in virtual spring implementation, it will be impossible to extract a net amount of energy from the virtual wall, regardless of trajectory or combination thereof.

3 Performance Improvement

Upon examining the coupled parameters expressed in (6), it follows that to maximize the virtual stiffness of a haptic device (without modification of the rendering algorithms), one should maximize the ratio of viscous damping to sampling period and the ratio of Coulomb friction to encoder resolution.

Concurrently, if the objective is to expand a haptic device's envelope of performance, one must juggle the consequences regarding the direct exploitation of the aforementioned variables. For instance, most haptic devices are mechanically designed with minimal viscous and Coulomb friction [3]. When combined with low inertial effects and gravity compensation, the user feels free space or zero impedance at the end effector when there is no applied actuation. For this reason, the commanded actuator force calculated in (5) is valid.

But if one were to increase b and/or f_c , the physical dynamics of the haptic device would no longer remain transparent to the human operator. This would necessitate the addition of negative virtual damping to maintain the illusion of unimpeded motion through free space. Incorporating any type of active virtual damping to the simulation – regardless of whether it's positive or negative – not only complicates the once-simple, linear calculation of commanded force, but also dictates the computation of the velocity state from a discrete position signal. Differentiation is notorious for amplifying high frequency noise [11] and essentially creates yet another, more turbulent source of energy leaks.

While the governing limitation on virtual-wall stiffness dictates the importance of scaling between viscous friction and sampling rate and between Coulomb friction and

encoder resolution, it should be clarified that these two ratios contribute independently to the upper bound. Hence, significantly increasing sampling rate might not lead to any extension of performance if the presiding limitation resides within the ratio of Coulomb friction and encoder resolution. Likewise, increasing the resolution of the encoder may not amount to any additional improvement without first increasing computer speeds.

In particular, the ratio of f_c and Δ is often relatively large due to the high quality of available encoders. While it has been proven that, with unsatisfactory position quantization, the ratio of coulomb friction to encoder resolution can be the limiting factor on maximum displayable stiffness by a haptic device [9], encoders with high-resolution (on the scale of micrometers) for a reasonable price are currently standard in the electronic market. With such accurate sensing now available, common haptic devices are not often limited by their encoder resolution.

In summary, modifying the dynamics of a device can compromise transparency, while minimizing the width of micro-scale encoder pulses seldom bears performance-improvement. Perhaps an expansion of achievable stiffness can be captured by a critical analysis of a haptic device's computational platform. If the virtual stiffness of a passive haptic simulation is indeed limited by the ratio of viscous damping to sampling rate, then optimizing the computational speed of the haptic rendering simulation can lead to improved performance.

3.1 Minimizing Sample Period

As aforementioned, several previous implementation techniques have sought to improve virtual wall performance by effectively minimizing the discrete time sample of one complete rendering cycle. One of the more elementary solutions previously

proposed: to simply obtain a more capable computer chip to handle the haptic simulation application. It is evident that semiconductor manufacturers are continually producing faster, more powerful processors, bolstering the rapid evolution of the personal computer. Conceivably, a herculean integrated circuit (IC) could escalate algorithmic rates of a sampled-data spring model to monumental frequencies, whittling the discrete sample period down to mere nanoseconds. However, while such a prodigious processor for a personal computer is not far from the present, and could quite possibly already exist, it would most certainly come at an hefty price.

Moreover, these PC processors are designed for general purpose, handling a multitude of tasks in one moment, relying on the operating system software to designate the type of operation and determine the order of its execution. An application-specific integrated circuit (ASIC), on the other hand, devotes all of its processing capabilities towards executing a single task from a pre-defined set of operations. Its logic gates are designed such that all decision making is handled completely on the chip, eliminating the necessity of a governing operating system interfacing altogether and thus, can process at ultra-fast speeds.

3.1.1 Decision Making in Hardware vs Software

To illuminate this advantage first consider the communication path between a rotational haptic manipulator and its controller when decision making is performed in software.

Figure 3.1: Typical Communication Path when Controlled in Software

Digital encoder signals from the device are first routed to a buffer located on the data acquisition hardware. The OS then launches a driver application interface (API) to place these signals into bit registers on the computer processor. From here, the application software reads the digital signals and performs the calculations necessary to determine the manipulator's current position. Additionally, if one does not employ a RT OS, the position update task frequently competes with other, unrelated software applications for precious processing resources, succumbing to the loose hierarchy of task execution (a particular problem for systems running Windows OS). The rate at which the manipulator's position state is updated is dependent upon the ability for the signal to pass through all of these stages successfully.

On the other hand, if the position signals were to be updated completely on hardware, as could be done with an ASIC, raw encoder signals would be passed directly through a specific series logic gates in order to obtain the manipulator's current position state; all intermediate software interfacing is removed from the communication path.

Figure 3.2: Optimized Communication Path when Controlled on Hardware

Data acquisition rates that were once on the scale of milliseconds are now reduced to a scale of microseconds, thus severely decreasing the duration of the ZOH and yielding a smaller variance between the sampled-data spring model and the continuous spring model.

3.1.2 Proposed Platform Solution

As its name suggests, every ASIC is designed for a specific use. As such, to obtain an ASIC for the purpose of communicating with a sophisticated haptic manipulator, one would devote several months, if not years to prototype the chip-logic design, not to mention the several thousands of dollars necessary to contract the manufacturing of such a device. It is of no consequence that ASICs are typically manufactured in mass-production quantities and are rarely the subject of prototype design.

Designers who seek the speed and efficiency of an ASIC who cannot afford such an abundance of time and money can turn to a FPGA. In contrast to the fixed functionality of an ASIC, an FPGA offers the luxury of reconfiguring the logic gates that govern the chips purpose; a designer compiles code defined in Hardware Description Language (HDL) to the device, configuring the circuitry such that it executes the desired tasks. Furthermore, if the designer chooses to redefine the functionality, one simply recompiles code to the device, flipping the gates in the logic circuitry to execute a new set of tasks. Thus, if one seeks to interface a dedicated, application-specific processor to an electromechanical device (for instance, a robotic manipulator) in order to boost data acquisition rates, an FPGA is ideal.

The functionality of an FPGA, while extremely fast in execution (on the scale of MHz [19]), is also somewhat limited in complexity. Access to the actual configuration of its logic gates also limits any on-board computations to Boolean and integer math operations. To operate a multi-dimensional, rotational haptic manipulator entirely from an FPGA, one would have to devise tactful and meticulous approximations of high order

operations to render even the most simple of haptic simulations. The forward kinematics transformation, for example, requires trigonometric operations on the joint space vector to represent the manipulator in Cartesian space.

To absolve this handicap, a RT execution target can be paired with the FPGA to handle any higher-order rendering. Specifically, it is proposed that an FPGA be utilized to acquire, process and command all digital information (encoder signals) while a separate, RT application that is executed on a generic, dedicated PC processor, handles the remaining operations necessary to calculate appropriate voltage commands to display a haptic interaction. Once the command signals are calculated, the RT OS can pass them to the FPGA device where they are sent to the actuators. In short, the FPGA acts as a high-speed data acquisition device, in addition to calculating the position state vector, and the RT OS renders the haptic environment control commands.

Using the described computational architecture, the benefits of computation in hardware and deterministic software are captured: namely, the efficiency of high-speed decision making on an FPGA and the coupled versatility of complex, deterministic virtual rendering in RT software. It is a platform that enhances computational loop rates of a haptic simulation and thus expands the envelope of performance of a haptic device.

4 Implementation

The previous chapter states that increasing the loop rate of a haptic simulation can increase the displayed stiffness of a passive virtual wall. To demonstrate this, the performance of a widely-used commercial haptic device, the PHANToM Premium 1.0 (SensAble Technologies) is analyzed when rendering an increasingly stiff unilateral spring on its standard computational platform. The results are then directly compared to the same haptic simulations when operating the device on a custom platform that features LabVIEW FPGA and LabVIEW Real-Time (National Instruments).

4.1 Haptic Device: The PHANToM Premium

Figure 4.1: PHANToM Premium 1.0 [21]

The PHANToM Premium 1.0 is one of several commercial, force-reflecting haptic manipulators manufactured by SensAble Technologies. It is a three degree-of-freedom (dof) robotic manipulator that employs three high resolution encoders to track a user's displacement of the end effector. Appropriate torque commands are then sent to three brushed DC motors coupled to each rotational axis, displaying a point-force vector at the user interface, which is typically a grasped stylus.

To examine the performance limitation expressed by the virtual wall passivity criterion derived in the previous chapter the following estimates of the governing parameters have been found for the PHANToM 1.0 [10]:

Table 4.1: PHANToM Premium Estimated Device Parameters

	Symbol	Value	Units
Viscous Damping Coefficient	b	.005	Ns/m
Sampling Period	T	.001	S
Coulomb Friction	f_c	.038	N
Encoder Resolution	Δ	29.1	Mm

Lending these values to Equation 6 reveals a theoretical maximum on the PHANToM's achievable stiffness of a virtual spring.

$$\frac{2b}{T} = 10 \text{ } N/m \quad (30)$$

$$\frac{2f_c}{\Delta} = 26,117 \text{ } N/m \quad (31)$$

$$K_{\max} = \min\left(\frac{2b}{T}, \frac{2f_c}{\Delta}\right) = 10 \text{ } N/m \quad (32)$$

It therefore stands that in its standard form and mode of operation the PHANToM is overwhelmingly limited by the ratio of viscous damping to sampling period. To maintain the transparency of free space, the viscous damping of the device will not be altered. Hence, to increase the bound on virtual spring stiffness of a passive virtual wall, the suggested strategy to shorten the sampling-delay is indeed merited.

But to forgo the default hardware platform and interface the PHANToM robot with an FPGA-based framework requires complete knowledge of the mechanical and electrical properties of the device, knowledge justifiably withheld by the original maker. Thus, a rigorous investigation of the PHANToM's architecture is necessary to properly

interface the robotic manipulator and successfully implement a virtual wall simulation with a custom operating system.

4.1.1 Impedance Based Implementation

As stated earlier, the PHANToM is an impedance-based device. If one considers an impedance-based approach to haptic rendering with a three dof rotational (3-R) robotic manipulator, the following block diagram depicts the basic computing methodology.

Figure 4.2: Impedance-Based Control Block Diagram

Rotational displacement of the manipulator is measured in joint space by the encoders, then transformed to a Cartesian displacement of the end effector (user position) with the forward kinematics matrix. This position state vector is subject to the simulated environment causality, resulting in a desired force to be commanded to the end effector. This force is then converted to a desired torque (to send to the motors) from the transpose of the Jacobian matrix. The actual commanded torque to the motors is the sum of desired torque and the torque resulting from the device's physical dynamics.

4.1.2 Forward Kinematics

When employing an impedance-based haptic simulation, it is necessary to explicitly and continuously express all geometrical and time-based properties of the force-reflecting manipulator's motion. With the derivation of the forward kinematics

transformation matrix, T_r^w , the position and orientation of the PHANToM end-effector is expressed as a function of its joint variables.

$$T_{Tool}^{World} = \begin{bmatrix} R(\theta) & \begin{matrix} p_{ox} \\ p_{oy} \\ p_{oz} \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (33)$$

It is important to note here that the forward kinematics transformation matrix is derived from the approach depicted in [22] which uses Denavit-Hartenberg parameters to express the transformations from each link frame. This technique assumes all axes of rotation are independent, i.e. that the manipulator is serial, which, by inspection, the PHANToM is not. Observe in Fig 4.3 that θ_2 and θ_3 share same axis of rotation.

Figure 4.3: Rotational Axes of PHANToM

It is permissible, however, to treat the PHANToM serially by decoupling the second and third axes of rotation in the forward kinematics derivation. The geometrical relationships between the former and the updated axes of rotation are detailed in Fig. 4.4 and (34) and (35).

Figure 4.4: Decoupling the PHANToM Joint Angles

$$\theta_2^* = \theta_2 - \frac{\pi}{2} \quad (34)$$

$$\theta_3^* = \theta_2^* + \theta_3 \quad (35)$$

A clear representation of the new joint variables is provided in Fig 4.5. This serial interpretation of each axis of rotation is actuated by a single, independent motor.

Figure 4.5: Serial Axes of Rotation

Adopting this serial representation, the following tool, world, and intermediate Cartesian-frame orientations are used to construct the D-H parameter table (Table 4.2).

Figure 4.6: Assigned Transformation Frames

Table 4.2: Denavit-Hartenberg Parameters

Frame	a_{i-1}	α_{i-1}	d_i	θ_i
W	0	0	0	0
0	0	0	0	0
1	0	$-\pi/2$	0	$-\pi/2$
2	0	0	L_2	θ_1
3	0	$-\pi/2$	0	θ_2
4	L_1	0	0	θ_3
5	0	0	0	$-\pi/2$
T	L_2	0	0	$\pi/2$

From the D-H notation of a forward frame transformation defined in [22],

$${}^{n-1}T_n = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (36)$$

the resulting forward kinematic map for the PHANToM is as follows:

$$T_T^W = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1)\sin(\theta_2^* + \theta_3^*) & \sin(\theta_1)\cos(\theta_2^* + \theta_3^*) & \sin(\theta_1)[L_1\cos(\theta_2^*) + L_2\sin(\theta_2^* + \theta_3^*)] \\ 0 & \cos(\theta_2^* + \theta_3^*) & \sin(\theta_2^* + \theta_3^*) & L_2 - L_2\cos(\theta_2^* + \theta_3^*) + L_1\sin(\theta_2^*) \\ -\sin(\theta_1) & -\cos(\theta_1)\sin(\theta_2^* + \theta_3^*) & \cos(\theta_1)\cos(\theta_2^* + \theta_3^*) & -L_1 + \cos(\theta_1)[L_1\cos(\theta_2^*) + L_2\sin(\theta_2^* + \theta_3^*)] \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (37)$$

Experimental verification of this forward kinematics derivation is provided in Appendix A.

4.1.3 Inverse Kinematics

Since an impedance-based approach to rendering is considered in this study the inverse-kinematics are not necessary for control. Despite this, it is good practice to include this derivation in any thorough manipulator kinematics study. For this 3-R manipulator, the set of joint angles, $(\theta_1, \theta_2, \theta_3)$, which move the end-effector to a desired position, $[p_{ox}, p_{oy}, p_{oz}]^T$, have been calculated in [23]. From a top view of the manipulator, θ_1 is determined by inspection.

Figure 4.7: PHANToM Top View [23]

$$\theta_1 = \arctan 2(p_{ox}, p_{oz} + l_1) \quad (38)$$

For θ_2 and θ_3 , consider the side view of the manipulator.

Figure 4.8: PHANToM Side View [23]

From inspection and by utilizing the law of cosines, the resulting angles are

$$\theta_2 = \cos^{-1}\left(\frac{l_1^2 + r^2 - l_2^2}{2l_1 r}\right) + a \tan 2(p_{oy} - l_2, d) \quad (39)$$

$$\theta_3 = \theta_2 + \cos^{-1}\left(\frac{l_1^2 + l_2^2 - r^2}{2l_1 l_2}\right) - \frac{\pi}{2} \quad (40)$$

where d and r are intermediate variables,

$$d = \sqrt{p_{ox}^2 + (p_{oz} + l_1)^2} \quad (41)$$

$$r = \sqrt{p_{ox}^2 + (p_{oy} - l_2)^2 + (p_{oz} + l_1)^2} . \quad (42)$$

4.1.4 Jacobian

The Jacobian matrix relates the manipulator's joint velocities to its Cartesian velocities [22].

$$\dot{x} = J(\theta)\ddot{\theta} \quad (43)$$

One can derive the Jacobian for a 3-R manipulator by calculating the first-order partial derivative of the position vector, $[p_{ox}, p_{oy}, p_{oz}]^T$ with respect to the joint vector, $[\theta_1, \theta_2, \theta_3]^T$.

$$J = \begin{bmatrix} \frac{\partial p_{ox}}{\partial \theta_1} & \frac{\partial p_{ox}}{\partial \theta_2} & \frac{\partial p_{ox}}{\partial \theta_3} \\ \frac{\partial p_{oy}}{\partial \theta_1} & \frac{\partial p_{oy}}{\partial \theta_2} & \frac{\partial p_{oy}}{\partial \theta_3} \\ \frac{\partial p_{oz}}{\partial \theta_1} & \frac{\partial p_{oz}}{\partial \theta_2} & \frac{\partial p_{oz}}{\partial \theta_3} \end{bmatrix} \quad (44)$$

Thus, the Jacobian for the PHANToM is calculated as follows:

$$J = \begin{bmatrix} \cos(\theta_1)[L_1 \cos(\theta_2) + L_2 \sin(\theta_3)] & -L_1 \sin(\theta_1) \sin(\theta_2) & L_2 \cos(\theta_1) \cos(\theta_3) \\ 0 & L_1 \cos(\theta_2) & L_2 \sin(\theta_3) \\ -\sin(\theta_1)[L_1 \cos(\theta_2) + L_2 \sin(\theta_3)] & -L_1 \cos(\theta_1) \sin(\theta_2) & L_2 \cos(\theta_1) \cos(\theta_3) \end{bmatrix} \quad (45)$$

The transpose of the Jacobian can then map Cartesian forces acting in the tool frame to equivalent joint torques.

$$\tau_{desired} = J(\theta)^T F_{desired} \quad (46)$$

4.1.5 Dynamics

In standard practice, it is necessary to know the dynamic equations of motion for a rotational robotic manipulator, in addition to its kinematics when commanding torque. The equations of motion for a rotational dynamic system can be expressed as follows:

$$M(\theta)\ddot{\theta} + V(\dot{\theta}, \theta)\dot{\theta} + G(\theta) = \tau_{dyn} \quad (47)$$

where $M(\theta)$ is the mass matrix, $V(\dot{\theta}, \theta)$ is the Coriolis and centrifugal terms matrix and $G(\theta)$ is the gravity matrix.

From a previous derivation of the PHANToM's dynamic equations in [23], one will observe that all three matrices have several terms with non-linear operations on the state vector. Yet it is reasonable to disregard the device dynamic torque and its associated nonlinear operations if one assumes low accelerations and velocities at the device

interface, and also recognizes that the PHANToM was specifically designed with low inertia, low friction, and physical gravity compensation of its rigid, lightweight linkage system.

One will find that the dynamic torque bears minimal weight on the torque commanded to the motors. In other words, the force inflicted by the device dynamics at the user interface will go undetected by the operator.

The work of O'Malley in [26] shows that users have poor performance in object detection when forces are saturated at $0.5N$.

Thus, if

$$F_{dyn} \leq 0.5N, \quad (48)$$

then

$$\tau_{commanded} = \tau_{desired} + \tau_{dynamic} \cong \tau_{desired}. \quad (49)$$

Refer to Appendix B for validation of this approximation.

Each stage of the block diagram depicted in Fig 4.2 is now fully understood. The input joint vector detects a change in the human operator's position. Subject to the causality of the simulated environment, a desired force vector is commanded to the end effector by means of the commanded torque vector; a motion input constitutes a force output.

It is now necessary to examine the electrical properties of the PHANToM in order to execute the impedance haptic algorithm digitally; i.e. to read the digital signals from the differential encoders and command voltages to the motors.

4.1.6 Electronic Properties

When one purchases a PHANToM robotic manipulator from SensAble Technologies, it comes in a plug and play fashion. One needs only to connect the hardware to a standard personal computer via a custom interface card and install the General Haptics Open Software Toolkit (GHOST) software to operate the haptic robot.

The C++ object-oriented development toolkit is intended to aid designers in modeling haptic environments through a hierarchal collection of geometric and spatial effects. The authors of the software state that “GHOST provides an abstraction that allows application designers to concentrate on the generation of haptic scenes...,” assuming that “application designers need not be concerned with low-level force and device issues” [24]. Consequently, any knowledge of the electromechanical makeup of the PHANToM is withheld by the maker.

Figure 4.9: PHANToM Commercial Package General Setup

The robotic device itself is connected to an amplifier box, the contents of which are left unknown to the user, although one can assume it contains pulse width modulation motor drive components and necessary signal conditioning electronics.

The conditioned signals coming from the amplifier box are passed through a custom, peripheral component interconnect (PCI) card that interfaces the PHANToM with a standard personal computer. This serial port card is made especially for the PHANToM, the signal mapping of which is also left unknown to the user.

Thus, in order to operate the force-reflecting robot on a custom hardware platform, the cable connecting the amplifier to the computer is intersected and the signals streaming through each wire are observed with an oscilloscope while the PHANToM runs in its default configuration. In this way, one “spies” on the PHANToM, determining the location, range and purpose of various digital and analog signals engaged in the robot’s operation. The wire mapping obtained from this scheme is recorded in Appendix C.

This blueprint of communication signals is the last step toward unraveling the hardware ambiguity of the PHANToM haptic interface. With a full and open architecture of the robotic manipulator’s mechanical and electrical properties it is now possible to implement a virtual environment on one’s own custom hardware platform.

4.2 FPGA-Based Hardware Platform

The FPGA chip to be interfaced with the PHANToM is housed within National Instrument’s R Series Intelligent DAQ Device, the PXI 7831R (See Appendix D.1 for the listed specifications for this device). The gates on this target are configured to acquire the digital signals from the differential encoders, compute the joint vector and command the analog voltages (computed by the RT application) to the PHANToM’s motors.

As previously mentioned, in order to define the behavior of the FPGA logic gates for custom I/O low-level operations, one must use HDL, specifically its most common

form: Very High Speed Integrated Circuit (VHSIC) HDL (VHDL). This FPGA-programming language typically has a considerable learning curve, and can therefore pose difficulties for those unfamiliar with its format.

However, National Instruments has developed an FPGA software module that allows a programmer to develop and compile custom VHDL commands to the chip using LabVIEW graphical programming software; any prior knowledge of VHDL is not necessary. The virtual instrument (.vi) code designed in LabVIEW FPGA to read the PHANToM's encoders and control its motors is provided in Appendix E.1.

The joint vector continuously updated by the FPGA is targeted by a separate, dedicated processor located on National Instrument's PXI 8176 Embedded Controller where the haptic rendering algorithm is executed in real time. The time-critical rendering loop is programmed using LabVIEW Real-Time Module in order to implement the haptic rendering sequence depicted in the block diagram in Fig. 4.2. The RT application retrieves the joint vector from the FPGA, calculates the appropriate voltage commands and sends them back to the FPGA. The LabVIEW code is deployed to the RT target through a TCP/IP network connection where the haptic simulation loop executes with ultimate priority. A separate Windows host computer maintains lateral communication with the RT application, providing a visual interface to the operator.

This front panel allows the user to designate virtual wall properties like wall location and spring stiffness, commands that are updated to the RT processing loop at a non-critical timing rate. The LabVIEW Real-Time .vi compiled to the RT target and the corresponding front panel (visual interface) located on the host machine are provided in Appendices E.2 and E.3, respectively.

Figure 4.10 : Custom Hardware Platform General Setup (adapted from [20])

A general description of the computational setup is depicted in Fig 4.10. The key highlights of this operating platform can be outlined as follows:

1. **The FPGA chip acts as a high-quality custom I/O device:** The pins and gates of the logic device are programmed to act as custom digital and analog inputs and outputs, bypassing any software calls and API's necessary for standard PC interfacing. It performs the low-level operations that read encoder signals and calculate the joint vector.
2. **No prior knowledge of VHDL:** LabVIEW FPGA module provides development of FPGA functionality in virtual instrumentation graphical programming.

3. RT application performs haptic rendering in deterministic fashion:

A separate processor is configured to run the RT application that handles the higher-order, time-critical tasks in the haptic rendering algorithm.

4. RT target accessed by host-computer application:

LabVIEW Real-Time module deploys the haptic rendering code to the RT target and manages the graphical interface on a separate host computer.

All of these advantages combine to produce an optimized, efficient communication path for the PHANToM haptic manipulator. When interfaced to the FPGA/RT platform, the PHANToM executes 3-D virtual wall simulation iterations at 20 kHz, significantly decreasing the sample period of its former operating platform by a factor of 20. Furthermore, because the algorithm is executed in deterministically, it is impervious to lags, interrupts, and crashes; the platform is not only optimal, it is reliable and safe.

5 Experiment

5.1 Setup

To demonstrate the significance of minimizing the control loop rate of the PHANToM haptic manipulator, a series of virtual wall performance tests are performed using the FPGA/RT hardware platform.

A planar, horizontal virtual wall is placed one centimeter below the end-effector's zero-position and is assigned a relatively small, initial virtual stiffness. To induce controlled, repeatable contact with the virtual floor, a small mass is bolted to the exterior link of the manipulator and released from an initial height offset of one centimeter. The weight of the mass is chosen such that it narrowly exceeds the intrinsic friction and gravity compensation of the device. This simulates a light user grip at the wall's threshold, the most challenging of scenarios of passive response for a virtual spring [10]; a heavy grip merely acts to add damping, assisting a passive interaction.

Figure 5.1: Drop Test Setup Diagram

The position of the end effector is recorded upon release, monitoring the second order response of the mass-spring system. Any sustained or growing oscillations

observed during the response are considered to be non-passive behaviors (energy leaks) of the virtual spring system. With each run, the virtual stiffness is increased and the end effector is released from the same initial position, making contact with the virtual wall at the same location

Because of the software-enabled kill-switch GHOST imposes on the PHANToM, it does not allow the user to test the limits of performance regarding exceptionally large virtual stiffness. Furthermore, due to the limited access to the virtual environment's application software commands, one cannot implement a virtual spring without the automatic coupling of virtual damping that aids in device stability.

The intention of this investigation is to observe the PHANToM's performance when pushed to the extremes of its passive region of operation without the aid of additional damping, a test not permissible when running GHOST. Thus, the drop-test experiment is conducted on the FPGA/RT platform at both 1 kHz and 20 kHz. It is assumed that the custom platform running at 1 kHz sufficiently emulates the performance of the default platform. It is noted that the non-deterministic nature of the Windows/GHOST platform is not captured; the suppression of lags and interrupts will only assist in performance.

5.2 Results

The following plots record the dynamic response of the physical-mass, virtual-spring system when the haptic control simulation is implemented at 1 kHz and 20 kHz. In Fig. 5.2-5.6, a visual critique is offered, comparing the virtual wall performances with increasing virtual stiffness when updated at the two platform rates.

(a)

(b)

Figure 5.2: Spring-mass drop test response with virtual stiffness of 1 N/m with loop rate of (a): 1 kHz, (b): 20 kHz

(a)

(b)

Figure 5.3: Spring-mass drop test response with virtual stiffness of 5 N/m with loop rate of (a): 1 kHz, (b): 20 kHz

(a)

(b)

Figure 5.4: Spring-mass drop test response with virtual stiffness of 10 N/m with loop rate of (a): 1 kHz, (b): 20 kHz

(a)

(b)

Figure 5.5: Spring-mass drop test response with virtual stiffness of 25 N/m with loop rate of (a): 1 kHz, (b): 20 kHz

(a)

(b)

Figure 5.6: Spring-mass drop test response with virtual stiffness of 50 N/m with loop rate of (a): 1 kHz, (b): 20 kHz

5.3 Observations

From the results of the virtual simulations, it is clear that in the lower range of stiffnesses between 1 N/m and 5 N/m both platforms seem to exhibit the same passive behavior, as seen in Fig. 5.2 and Fig. 5.3. The end-effector settles on the virtual surface with a smooth, stable position.

However, when the desired virtual stiffness reaches 10 N/m in Fig. 5.4, small yet sustained oscillations begin to appear at the wall threshold when updated at 1 kHz while the 20 kHz simulation maintains stable settling behavior. The oscillations seen in Fig. 5.2(a) at the virtual surface would be experienced by a human user with a light grasp as faint vibrations, the magnitude of which would increase as the virtual stiffness is increased further beyond the passive limit. These oscillations are heightened when virtual stiffness is increased to 25 N/m in Fig. 5.5(a).

The consequences of operating beyond the passive region of a haptic simulation become apparent in Fig. 5.6. At 50 N/m on the 1 kHz platform, the growing oscillations are too great in magnitude for the device to overcome with intrinsic damping, thus rendering the system unstable. Meantime, the 20 kHz simulation maintains passive surface behavior.

The remaining plots are the drop test results obtained for the 20 kHz simulations when the virtual stiffness is set beyond the stable limit of the 1 kHz platform.

Figure 5.7: Spring-mass response with virtual stiffness of 100 N/m at 20 kHz

Figure 5.8: Spring-mass response with virtual stiffness of 200 N/m at 20 kHz

Figure 5.9: Spring-mass response with virtual stiffness of 300 N/m at 20 kHz

Figure 5.10: Spring-mass response with virtual stiffness of 600 N/m at 20 kHz

One will observe in Fig. 5.8 that it is not until the virtual stiffness reaches 200 N/m that the FPGA/RT platform reveals even the slightest of sustained oscillations. It is worthy of noting that in Fig. 5.7-5.10, the virtual wall system is never rendered unstable, even at 600 N/m, a virtual stiffness far beyond its passive region.

Because of exceptionally faster loop rate, the performance of the FPGA-based platform far exceeds that of the Windows/GHOST platform, rendering passive haptic interactions with higher virtual stiffness.

Conclusions

The challenge of displaying stiff virtual walls in a haptic simulation was the inspiration for this work. A passivity analysis was conducted to examine the factors that can affect the fidelity of virtual wall implementation. To overcome the inevitable energy leaks resulting from discrete simulations, it was proposed that the servo rate of the haptic interaction be increased.

This was done by integrating an FPGA logic device with an RT OS, using hardware and software resources provided by National Instruments. This custom computational platform was interfaced to the PHANToM Premium commercial haptic device, resulting in a complete and open architecture of the haptic manipulator. The performance of the device was then observed when operating on the FPGA/RT hardware platform. The results affirmed increased stiffness in the PHANToM's passive region of operation. The increase in displayed stiffness is regarded as an improvement in haptic performance when rendering virtual walls.

References

- [1] K. Salisbury, C. Francois, F. Barbagli, "Haptic rendering: Introductory concepts," *IEEE Computer Graphics and Applications*, vol. 24, no. 2, March/April 2004, pp. 24-32.
- [2] K. Kuckenbecker, G. Niemeyer, "Canceling induced Master Motion in force-reflecting teleoperation," in *Proc. ASME IMECE*, vol. DSC-vol. 73, Anaheim, CA, Nov. 2004, pp. 1091-1098.
- [3] R. Ellis, O. Ismaeil, M. Lipsett, "Design and evalutaion of a high-performace haptic interface." *Robotica*, vol. 14, no. 3, pp. 321-327, 1996.
- [4] M. O'Malley, M. Goldfarb, "The effect of force saturation on the haptic perception of detail," *IEEE/ASME Trans. Mech.*, vol. 7, no. 3, pp. 280-288, Sept. 2002.
- [5] L. Rosenberg, B. Adelstein, "Perceptual Decomposition of Virtual Haptic Surfaces," in *Proc. IEEE Virtual Reality Symp.* 1993, pp. 43-53.
- [6] J. Colgate, P. Grafin, M. Stanley, and G. Schenkel, "Implementation of stiff virtual walls in force-reflecting interfaces," in *Proc. IEEE Virtual Reality Symp.*, 1993, pp. 202-208.
- [7] H. Kazerooni, "Human induced instability in haptic interfaces," in *Proc. ASME Wint. Ann. Mtg.*, vol. DSC-vol. 55, no. 2, 1993, pp. 851-856.
- [8] B. Gillespie and M. Cutkosky, "Stable user-specific rendering of the virtual wall," in *Proc. ASME IMECE*, vol. DSC-vol. 58, Atlanta, GA, Nov. 1996, pp. 397-406.
- [9] J. J. Abbott and A. M. Okamura, "Effects of position quantization and sampling rate on virtual-wall passivity," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 952-964, Oct. 2005.
- [11] J. Colgate, J. Brown, "Factors affecting the z-width of a haptic display," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1994, pp. 3205-3210.
- [12] B. Miller, J. Colgate, and R. Freeman, "On the role of dissipation in haptic systems," *IEEE Trans. Robot.*, vol. 20, no. 4, pp. 768-771, Aug. 2004.

- [13] X. Shen and M. Goldfarb, "Leveraging the open-loop stiffness of pneumatic actuators to enable high-stiffness surface simulation in impedance-type haptic interfaces," *Proc. ASME IMECE*, vol. DSC-vol. 74, Orlando, FL, Nov. 2005, pp. 1421-1428.
- [14] B. Hannaford and J. Ryu, "Time-domain passivity control of haptic interfaces," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seoul, Korea, May 2001, pp. 1863-1869.
- [15] V. Hayward and O. Astley, "Performance measures for haptic interfaces," in *Proc. 1996 Robotics Research: 7th Int'l Symp.*, 1996, pp. 195-207.
- [16] M. Mahvash and V. Hayward, "High-fidelity passive force reflecting virtual environments," *IEEE Trans. Robot.*, vol. 21, no. 1, pp. 38-46, Feb. 2005.
- [17] K. Kavi and S. Yang, "Real-Time systems design methodologies: An introduction and survey," *Journal of Systems and Software*, vol. 18, no. 1, Apr. 1992, p 85-99.
- [18] P. V. O'Neil, *Advanced Engineering Mathematics*. Pacific Grove: Brooks/Cole, 2003.
- [19] National Instruments, "Introduction to LabVIEW FPGA," *LabVIEW 8 FPGA Module Training*, National Instruments Developer Zone, PowerPoint, 3 Jul. 2006, <http://zone.ni.com/devzone/cda/tut/p/id/3555#toc0/lesson_0_intro_to_lv_fpga_8.ppt>.
- [20] National Instruments, "LabVIEW FPGA Basics," *LabVIEW 8 FPGA Module Training*, National Instruments Developer Zone, PowerPoint, 3 Jul. 2006, http://zone.ni.com/devzone/cda/tut/p/id/3555#toc0/lesson_1_lv_fpga_basics.ppt.
- [21] PHANToM Premium 1.0, Sensable Technologies, image, 8 Aug. 2006, <<http://www.sensable.com/phantom-premium-1-0.htm>>.
- [22] J. J. Craig, *Introduction to Robotics: Mechanics and Control - 3rd Edition*. New Jersey: Pearson Prentice Hall, 2005.
- [23] M. Cavusoglu, D. Feygin, F. Tendick, "A critical study of the mechanical and electrical properties of the PHANToM haptic interface and improvements for high-performance control," *Presence*, vol. 11, no. 6, Dec. 2002, p 555-68.
- [24] "GHOST SDK API Reference Version 4.0," *GHOST SDK Installation CD ROM* SensAble Technologies, electronic reference manual, ch.2, sec. 1, 2002.

- [25] Niemeyer, G., "Using Wave Variables in Time Delayed Force Reflecting Teleoperation," Doctorate of Philosophy Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1996.
- [26] Kilchemnan(O'Malley), M. "Implications of Machine Design for the Haptic Display of Detail," Doctorate of Philosophy Dissertation, Vanderbilt University, Nashville, TN, August 2001.

Appendix A:

Forward Kinematics Transformation Matrix Verification

To verify the legitimacy of the derived forward kinematics matrix, the joint vector is passed through the calculated transformation matrix and compared to the calculated Cartesian position state vector returned by GHOST.

For the verification test, the end effector is continuously displaced within its 3-D workspace in a random fashion. The corresponding joint space vector is obtained from GHOST. The joint space vector is passed through the derived forward kinematics matrix found in Eq. (9). These Cartesian coordinates are directly compared to those calculated by GHOST, as seen in Fig. A.1. The m-file created to display this comparison is also provided.

```
data=load...
('C:\Documents and Settings\Owner\My Documents\PHANTOM\forkindata.txt');
%Columns 9-11 are X,Y,Z coordinates calculated by GHOST, respectively
%Columns 18-20 are theta 1,2,3 joint angles recorded by GHOST, in radians
t=data((1:2000),1);
X_GHOST=-data((2501:4500),9)/10;
Y_GHOST=data((2501:4500),10)/10;
Z_GHOST=data((2501:4500),11)/10;

%Ghost_Pos=[X_P';Y_P';Z_P'];
th1=data((2501:4500),18);
th2=data((2501:4500),19);
th3=data((2501:4500),20)-th2;

%Link Lengths from Cava [mm]
L1=.215/1.5*1000;
L2=.170/1.5*1000;

%Cartesian coordinates calculated from MY forward kinematics transformation
X_Calc=(sin(th1).*(L1.*cos(th2)+L2.*sin(th2+th3)))/10;
Y_Calc=(L2-L2*cos(th2+th3)+L1*sin(th2))/10;
Z_Calc=(-L1+cos(th1).*(L1.*cos(th2)+L2.*sin(th2+th3)))/10;

X_diff=abs(X_GHOST-X_Calc);
Y_diff=abs(Y_GHOST-Y_Calc);
Z_diff=abs(Z_GHOST-Y_Calc);

figure
subplot(3,1,1);plot(t',X_Calc,'r',t',X_GHOST,'b');axis([0 10 -20 20]);...
    title('EndEffector Coordinates');ylabel('X [cm]');
subplot(3,1,2);plot(t',Y_Calc,'r',t',Y_GHOST,'b');axis([1 10 -20 20]);...
    ylabel('Y [cm]');
subplot(3,1,3);plot(t',Z_Calc,'r',t',Z_GHOST,'b');axis([1 10 -20 20]);...
    ylabel('Z [cm]');xlabel('time [s]')
```

Figure A.1: Forward Kinematics Verification Simulation Results

Appendix B:

Dynamic Torque Assumption

To verify that the torque due to the device dynamics is negligible, the resulting end-effector force is calculated when the PHANToM is moved around in free space. If the magnitude of the calculated force in each Cartesian axis is less than .5N, then the negligible dynamics assumption is considered valid [26]. The m-files and command window employed to verify this are included.

```
data=dlmread...
('C:\Documents and Settings\Owner\My Documents\PHANTOM\phantomdynamicdata.txt');
th1=data(:,1);
th2=data(:,2);
th3=data(:,3);
th1dot=data(:,5);
th2dot=data(:,6);
th3dot=data(:,7);
th1dbldot=data(:,8);
th2dbldot=data(:,9);
th3dbldot=data(:,10);

%PHANTOM Parameters from [23]
l1 = .215/1.5; l2 = .170/1.5; l3 = .0325/1.5; l5 = -.0368/1.5; l6 = .0527/1.5;
ma = .0202; Iaxx = .4864*10^(-4); Iayy = .0018*10^(-4); Iazz = .4864*10^(-4);
mc = .0249; Icxx = .959*10^(-4); Icy = .959*10^(-4); Icz = .0051*10^(-4);
mbe = .2359; Ibexx = 11.09*10^(-4); Ibey = 10.06*10^(-4); Ibez = .591*10^(-4);
mdf = .1906; Idfx = 7.11*10^(-4); Idfy = .629*10^(-4); Idfz = 6.246*10^(-4);
Ibasey = 11.87*10^(-4);
g = 9.81;

%Inertial, Coriolis&Centrifugal and Gravity Matrix Elements
M11 = ((1/8)*(4*Iayy + 4*Iazz + 8*Ibasey + 4*Ibey + 4*Ibez + 4*Icy + 4*Icz...
+ 4*Idfy + 4*Idfz + 4*ma*(l1^2) + ma*(l2^2) + (l1^2)*mc + 4*(l3^2)*mc)...
+(1/8)*(4*Ibey - 4*Ibez + 4*Icy - 4*Icz + l1^2*(4*ma + mc))*cos(2*th2)...
+ (1/8)*(4*Iayy - 4*Iazz + 4*Idfy - 4*Idfz - l2^2*ma - 4*l3^2*mc)*cos(2*th3)...
+ l1*(l2*ma + l3*mc)*cos(th2).*sin(th3));
M12 = zeros(size(M11));
M13 = zeros(size(M11));
M21 = zeros(size(M11));
M22 = (1/4)*(4*(Ibexx + Icxx + l1^2*ma) + l1^2*mc)*th2dbldot;
M23 = (-1/2)*l1*(l2*ma + l3*mc)*sin(th2 - th3);
M31 = zeros(size(M11));
M32 = (-1/2)*l1*(l2*ma + l3*mc)*sin(th2 - th3).*th2dbldot;
M33 = ones(1382,1)*(1/4)*(4*Iaxx + 4*Idfx + l2^2*ma + 4*l3^2*mc);
V11 = (1/8)*(-2.*sin(th2).*((4*Ibey - 4*Ibez + 4*Icy - 4*Icz + 4*l1^2*ma + l1^2*mc)...
.*cos(th2)+2*l1*(l2*ma + l3*mc).*sin(th3)).*th2dot+2*cos(th3).*(2*l1*(l2*ma + ...
l3*mc).*cos(th2)+(-4*Iayy + 4*Iazz - 4*Idfy + 4*Idfz + l2^2*ma + l3^2*mc).*...
sin(th3)).*th3dot);
V21 = (1/8)*((4*Ibey - 4*Ibez + 4*Icy - 4*Icz + l1^2*(4*ma + mc))*sin(2*th2)...
+ 4*l1*(l2*ma + l3*mc)*sin(th2).*sin(th3)).*th1dot;
V22 = zeros(size(M11));
V23 = (1/2)*l1*(l2*ma + l3*mc)*cos(th2 - th3).*th3dot;
V31 = (1/8)*(-4*l1*(l2*ma + l3*mc)*cos(th2).*cos(th3) - (-4*Iayy + 4*Iazz - 4*Idfy...
+ 4*Idfz + l2^2*ma + 4*l3^2*mc)*sin(2*th3)).*th1dot;
V32 = (1/2)*l1*(l2*ma + l3*mc)*cos(th2 - th3).*th2dot;
V12 = -V21;
V13 = -V31;
V33 = zeros(size(M11));
G1 = zeros(size(M11));
G2 = (1/2)*g*(2*l1*ma + 2*l5*mbe + l1*mc)*cos(th2);
G3 = (1/2)*g*(l2*ma + 2*l3*mc - 2*l6*mdf)*sin(th3);

%Dynamic Torque Vector
tau1=M11.*th1dbldot+V11.*th1dot+V12.*th2dot+V13.*th3dot;
tau2=M22.*th2dbldot+M23.*th3dbldot+V21.*th1dot+V23.*th3dot+G2;
```

December 6, 2006

2:18:11 AM

$$\tau_{u3} = M_{32} \cdot \ddot{\theta}_2 + M_{33} \cdot \ddot{\theta}_3 + V_{31} \cdot \dot{\theta}_1 + V_{32} \cdot \dot{\theta}_2 + G_3;$$

%Inverse Jacobian Transpose (retrieved from inversejacobiantranspose.m)

```
J11=(conj(l1.*cos(th2)).*conj(l2.*cos(th1).*cos(th3))+conj(l1.*cos(th1).*sin(th2))...
.*conj(l2.*sin(th3)))./(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos...
(th2)).*conj(l2*cos(th1).*cos(th3))+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*...
.*conj(l1*cos(th1).*sin(th2)).*conj(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2))...
.*conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1)...
.*cos(th3)).*conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J12=-conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3))./(conj(cos(th1)...
.* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)).*conj(l2*cos(th1).*cos(th3))...
+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1.*cos(th1).*sin(th2)).*conj...
(l2.*sin(th3))+conj(l1.*sin(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+l2*...
sin(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1).* (l1...
*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J13=conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1.*cos(th2))./(conj(cos(th1)...
.* (l1*cos(th2)+l2*sin(th3))).*conj(l1.*cos(th2)).*conj(l2.*cos(th1).*cos(th3)...
))+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1.*cos(th1).*sin(th2)).*...
conj(l2.*sin(th3))+conj(l1.*sin(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+...
l2.*sin(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1)...
.* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J21=-conj(l2*cos(th1).*cos(th3)).*(-conj(l1*sin(th1).*sin(th2))+conj(l1*cos(th1).*...
sin(th2)))./(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)).*conj...
(l2*cos(th1).*cos(th3))+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(...
th1).*sin(th2)).*conj(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2)).*conj(sin(th1).*...
(l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj...
(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J22=conj(l2*cos(th1).*cos(th3)).*(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3)))+conj...
(sin(th1).* (l1*cos(th2)+l2*sin(th3)))./(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3)...
)).*conj(l1*cos(th2)).*conj(l2*cos(th1).*cos(th3))+conj(cos(th1).* (l1*cos(th2)...
+l2*sin(th3))).*conj(l1*cos(th1).*sin(th2)).*conj(l2.*sin(th3))+conj(l1*sin...
(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3)...
)+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj...
(l1*cos(th2)));
J23=(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th1).*sin(th2))+conj...
(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*sin(th1).*sin(th2)))./(conj(cos...
(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)).*conj(l2*cos(th1).*cos(th3))...
+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th1).*sin(th2)).*conj...
(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+l2*sin...
(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1).* (l1*cos...
(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J31=-(conj(l1*sin(th1).*sin(th2)).*conj(l2.*sin(th3))+conj(l1*cos(th2)).*conj(l2*...
cos(th1).*cos(th3)))./(conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos...
(th2)).*conj(l2*cos(th1).*cos(th3))+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*...
.*conj(l1*cos(th1).*sin(th2)).*conj(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2)).*...
conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1)...
.*cos(th3)).*conj(sin(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J32=-conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l2.*sin(th3))./(conj(cos(th1)...
.* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)).*conj(l2*cos(th1).*cos(th3))...
+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th1).*sin(th2)).*conj...
(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+l2*sin...
(th3))).*conj(l2.*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1).* (l1*...
cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
J33=conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2))./(conj(cos(th1)...
.* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)).*conj(l2*cos(th1).*cos(th3))...
+conj(cos(th1).* (l1*cos(th2)+l2*sin(th3))).*conj(l1*cos(th1).*sin(th2)).*conj...
(l2.*sin(th3))+conj(l1*sin(th1).*sin(th2)).*conj(sin(th1).* (l1*cos(th2)+l2*sin...
```

```
(th3))).*conj(l2*sin(th3))+conj(l2*cos(th1).*cos(th3)).*conj(sin(th1).*(l1*...  
cos(th2)+l2*sin(th3))).*conj(l1*cos(th2)));
```

```
%Force experienced at End Effector
```

```
Fx=J11.*tau1+J12.*tau2+J13.*tau3;
```

```
Fy=J21.*tau1+J22.*tau2+J23.*tau3;
```

```
Fz=J31.*tau1+J32.*tau2+J33.*tau3;
```

```
%Max Force experienced at End Effector
```

```
Fx_max=max(Fx)
```

```
Fy_max=max(Fy)
```

```
Fz_max=max(Fz)
```

```
syms th1; syms th2; syms th3; syms l1; syms l2;
```

```
J=[cos(th1)*(l1*cos(th2)+l2*sin(th3)) -l1*sin(th1)*sin(th2) l2*cos(th1)*cos(th3);  
    th1*0 l1*cos(th2) l2*sin(th3);  
    -sin(th1)*(l1*cos(th2)+l2*sin(th3)) -l1*cos(th1)*sin(th2) l2*cos(th1)*cos(th3)];  
Jt_inv=(J')^(-1);
```

```
J11=Jt_inv(1,1);  
J12=Jt_inv(1,2);  
J13=Jt_inv(1,3);  
J21=Jt_inv(2,1);  
J22=Jt_inv(2,2);  
J23=Jt_inv(2,3);  
J31=Jt_inv(3,1);  
J32=Jt_inv(3,2);  
J33=Jt_inv(3,3);
```

Using Toolbox Path Cache. Type "help toolbox_path_cache" for more info.

To get started, select "MATLAB Help" from the Help menu.

```
>> inversejacobiantranspose  
>> phantom_negligible_dynamics
```

```
Fx_max =  
  
    0.3206
```

```
Fy_max =  
  
   -0.0293
```

```
Fz_max =  
  
    0.1353
```

```
>>
```

Appendix C:

PHANToM Communication Wire Map

The following table is the wire mapping configuration that interfaces the PHANToM Premium 1.0 to the FPGA device, NI PXI-7831R.

37 Pin Phantom	Function	68 Pin FPGA	Function
1	motor 1	48	AO 7
2	motor 2	50	AO 5
3	motor 3	49	AO 6
4			
5			
6			
7	GND	21	AOGND 0
8	ENC 1A	36	DIO 0
9			
10	ENC 3A	39	DIO 3
11			
12	ENC 2A	41	DIO 5
13			
14	5V	35	5V
15			
16			
17			
18	watchdog	42	DIO 6
19	stylus switch		
20	GND	21	AOGND 0
21	GND	21	AOGND 0
22	GND	21	AOGND 0
23			
24			
25			
26	5V	35	5V
27	ENC 1B	37	DIO 1
28	ENC 3B	38	DIO 2
29			
30	ENC 2B	40	DIO 4
31			
32			
33			
34			
35			
36			
37			

Appendix D:

Platform Hardware

The following page lists the specifications of the FPGA device employed for custom data acquisition and low-level computation.

R Series Intelligent DAQ with Onboard Processing

NI R Series *NEW!*

- Intelligent DAQ devices with programmable onboard decision making and digital signal processing
- Precise I/O synchronization, timing, triggering, and control with 25 ns resolution configured with the LabVIEW FPGA Module
- 1M or 3M gate FPGA chips
- Up to 8 analog input channels, 16-bit resolution, 200 kHz simultaneous-sampling rate
- Up to 8 analog output channels, 16-bit resolution, 1 MHz simultaneous update rate, ± 10 V
- Up to 160 high-speed digital lines configurable at rates up to 40 MHz for input, output, counter, or custom functionality
- Implement custom control logic or digital communication protocols

Operating Systems

- Windows 2000/XP
- LabVIEW Real-Time (ETS and RTX)

Recommended Software

- LabVIEW
- LabVIEW Real-Time Module
- LabVIEW FPGA Module
- LabVIEW code compiler for FPGAs
- Emulated debugging mode

Driver Software (included)

- NI-RIO

Calibration Certificate Available

Product	Bus	FPGA Size (gates)	Input/Output Resolution (bits)	Analog Inputs	Max Sampling Rate (kHz/ch)	Analog Input Range (V)	Analog Outputs	Max Update Rate (MHz/ch)	Digital I/O	Triggering (configurable per channel)
Multifunction										
NI 7830R	PCI/PXI	1M	16	4	200	± 10	4	1	56	Analog/digital
NI 7831R	PCI/PXI	1M	16	8	200	± 10	8	1	96	Analog/digital
NI 7833R	PCI/PXI	3M	16	8	200	± 10	8	1	96	Analog/digital
Digital										
NI 7811R	PCI/PXI	1M	—	—	—	—	—	—	160	Digital
NI 7813R	PCI/PXI	3M	—	—	—	—	—	—	160	Digital

Table 1. R Series Selection Guide

Overview

National Instruments R Series devices are equipped with reconfigurable input/output (RIO) hardware technology featuring onboard digital signal processing. The core of RIO is an FPGA (field-programmable gate array), which is configurable with the National Instruments LabVIEW FPGA Module. These intelligent DAQ devices literally rewire their internal gate array circuitry based on your LabVIEW FPGA program. You can customize the behavior of the device, providing capabilities such as:

- Complete control over synchronization and timing of all operations with 25 ns resolution
- User-defined, onboard decision making and triggering with loop rates up to 40 MHz
- Ability to individually configure digital lines as input, output, counter/timer, pulse-width modulator (PWM), encoder inputs, or user-defined communication protocols
- Simultaneous analog input at up to 200 kHz and simultaneous analog output at up to 1 MHz
- Multirate analog and discrete control

Because of these capabilities, R Series devices and the LabVIEW FPGA Module extend the National Instruments platform and can address applications including:

- High-speed, deterministic analog or discrete control
- Hardware-in-the-loop (HIL) simulation
- Rapid control prototyping (RCP)

Key Features

Through programming in LabVIEW FPGA, you can control each of the I/O signal lines independently or synchronize a line with other lines. You can configure the digital I/O lines as custom counter/timers, PWM channels, or ports for user-defined protocols. All NI 783x R Series devices have dedicated analog-to-digital converters/digital-to-analog converters on every analog I/O channel, making it possible to sample/update all channels simultaneously or at different rates. You can sample every analog input channel on an R Series device simultaneously at rates up to 200 kHz, and you can program every analog output on an R Series device to update simultaneously at rates up to 1 MHz. You can also store your user-defined LabVIEW FPGA configuration in flash memory on any R Series device providing for automatic loading and/or execution of the user program at power up.

Appendix E:

Platform Software

The virtual instrumentation code implemented in LabVIEW FPGA and LabVIEW RealTime is presented. The front panel of the RT application that serves as the graphical user interface is also provided.

PhantomDeviceIO_forRT.vi

C:\Documents and Settings\Owner\Desktop\updated phantom vi's\

PhantomDeviceIO_forRT.vi

Last modified on 12/7/2006 at 1:19 AM

Printed on 12/7/2006 at 10:51 AM

Virtual Room.vi

C:\Documents and Settings\Owner\Desktop\updated phantom vi's\My New VI's\Virtual Room.vi

Last modified on 12/7/2006 at 9:49 AM

Printed on 12/7/2006 at 11:00 AM

Virtual Room.vi

C:\Documents and Settings\Owner\Desktop\updated phantom vi's\My New VI's\

Virtual Room.vi

Last modified on 12/7/2006 at 9:49 AM

Printed on 12/7/2006 at 11:18 AM

Forward Kinematics.vi

C:\Documents and Settings\Owner\Desktop\updated phantom vi's\My New VI's\

Forward Kinematics.vi

Last modified on 12/7/2006 at 2:28 AM

Printed on 12/7/2006 at 11:21 AM

Torque to Voltage.vi

C:\Documents and Settings\Owner\Desktop\updated phantom vi's\My New VI's\

Torque to Voltage.vi

Last modified on 12/7/2006 at 3:50 AM

Printed on 12/7/2006 at 11:23 AM