CONTROL ELECTRONICS AND HYBRID DYNAMIC SYSTEM-BASED API FOR A 6-DOF DESKTOP HAPTIC INTERFACE

S.E. Salcudean†, R. Six†, R. Barman‡, S. Kingdon‡, I. Chau†, D. Murray‡ and M. Steenburgh‡
† Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, V6T 1Z4, Canada
tims@ece.ubc.ca

‡Point Grey Research
101-1847 W. Broadway
Vancouver, BC, V6J 1Y6, Canada
rodb@ptgrey.com

ABSTRACT

A six-degree-of-freedom desktop magnetically levitated haptic interface has been developed by the authors. Its electromechanical design is described in (Salcudean and Parker, 1997). In this paper, aspects of electronic hardware architecture and the control of actuator currents are discussed.

To program this device, a new low level applications programming interface (API) that models the haptic interface as a hybrid dynamic system is proposed. The user can define a finite state machine in which every state is a device impedance. State transitions occur upon the satisfaction of linear inequalities in terms of the device location, velocity and force. Examples of the use of such hybrid dynamic systems to produce haptic effects are given.

INTRODUCTION

The development of computer-user interfaces invariably requires a significant amount of experimentation. Haptic interfaces are no exception. Devices are needed to determine performance specifications, to develop applications and to evaluate their benefit. If poorly performing devices are used, such as conventional robots, applications using haptics interfaces cannot be properly developed and evaluated. However, the generally accepted specifications of haptic devices impose spatial acceleration and resolution figures that are difficult to obtain with conventional electromechanical designs. Therefore, some researchers have resorted to using devices with few degrees of freedom and developing appropriate metaphors, such as maneuvering a 2-D or 3-D "point" (see, for example, (Massie and Salisbury, 1994)). As an alternative, a high performance desk-top magnetically levitated haptic interface called PowerMouse has been developed (Salcudean and Parker, 1997). The device has six degrees of freedom, high acceleration, but limited motion range. Other magnetically levitated haptic interfaces have been developed and reported, with the most advanced demonstration of interaction with virtual environments being presented in (Berkelman and Hollis, 1999). However, these designs are not easily portable, require significant space and are awkward to interface to.

Design improvements to the PowerMouse device have been implemented and are briefly discussed in this paper. A new approach to the design of haptic APIs is also presented. In this approach, the haptic interface emulates a finite state machine in which every state is a device impedance. State transitions occur on the satisfaction of linear inequalities in terms of the device location, velocity and force.

The paper starts with a brief review of the PowerMouse electromechanical design. An overview of the electronic hardware design is given next. Experiments with different coil driving approaches using pulse width modulation have been carried out and are discussed. The structure of the finite state machine emulated by the haptic interface and accessible to the user via the device API is described next, with examples of some of the haptic effects that can be generated by such structure. Finally, conclusions are

presented.

POWERMOUSE ELECTROMECHANICAL DESIGN

The PowerMouse electromechanical design is summarized in this section. As seen in Figure 1, the device has a handle attached to a cubic "flotor" structure with the flat coils of six Lorentz actuators embedded in its faces. Twenty-four magnets on a stator structure generate the six magnetic fields that cross the coils. The wide magnetic gaps of the Lorentz actuators allow six-degree-of-freedom (6-DOF) motion of the flotor, with a motion range of $\pm$ 3 mm and $\pm 5°$ from a nominal center.

An optical 6-DOF sensor detects the flotor motion with a resolution of approximately 10 microns and 0.05 degrees. The sensor uses three LED-generated infrared light planes projected in sequence on three linear position sensing diodes (PSDs), mounted as an equilateral triangle on the PowerMouse printed circuit (PC) board. Each light plane crosses two PSDs. Thus six light-plane intersections with PSDs are obtained, allowing for the solution of the handle location using a direct kinematics computation.

The peak force of the device is 34 N, and the maximum continuous force is of the order of 16 N. The actuators have been optimized to maximize the force to power consumption ratio. It has been experimentally verified that, a result of the optimized actuators, the device operates far below its limits suggesting that a larger workspace could be obtained by increasing the magnetic gaps.

POWERMOUSE HARDWARE

A new controller board has been designed and successfully tested since the initial PowerMouse design was published. Its main features and operation are described in this section.

The heart of the PowerMouse control electronics is a Motorola MC5307 Coldfire processor as shows in Figure 2. The MCF5307 is a high-performance, low-cost derivative of Motorola's popular 68000 processors. It runs at 90 MHz and has a 70 MIPS Drystone rating. The processor has 16 MB of fast SDRAM main memory and 2 MB of flash non-volatile memory. Machine code and data is stored in a compressed format inside the flash memory and expanded into RAM at boot time for execution.

The Coldfire processor bus is shared with a Xilinx Spartan field programmable gate array (FPGA). The FPGA generates PWM modulation signals for driving the flotor coils, provides control signals for the light plane LEDs and interfaces with the two 16-bit analog-to-digital converters that measure PSD output currents. Six button inputs and

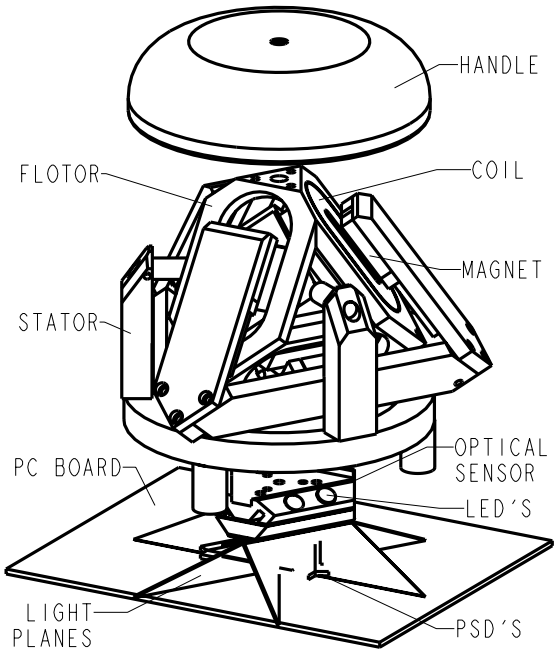

Figure 1. Schematic of the PowerMouse mechanical design.

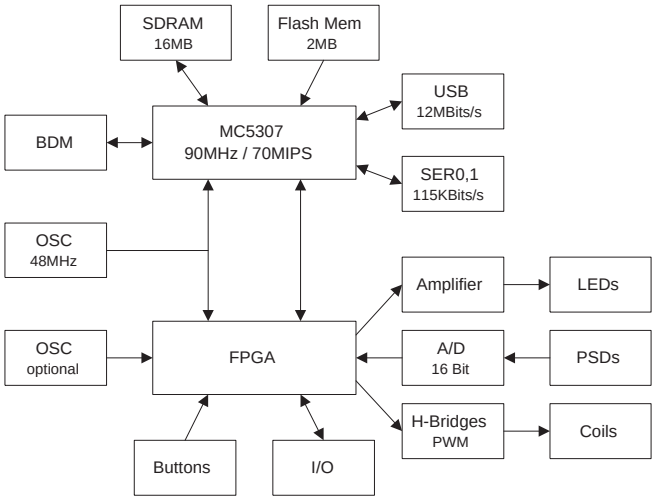

Figure 2. Schematic of the PowerMouse control electronics.

thirty additional I/O lines allow one to customize the device in various ways, *e.g.*, by the addition of a second clock, buttons, interface to force/torque sensor, etc. The use of an FPGA has allowed different optical sensing and coil driving algorithms to be easily tested without hardware redesign.

The PowerMouse communicates with a host PC using a 12 Mbit/s universal serial bus interface (USB). USB is an ideal choice for peripherals requiring inexpensive, low-

latency, high-bandwidth connections. Device drivers transparently load on the host computer when the USB connection is plugged in. Two 115Kb serial ports and the background debug mode BDM interface of the processor can be used for communication and development.

The PowerMouse is designed to operate with a 18-48V power supply and accommodates a H-bridge driving circuitry which is capable of driving 3A (6A peak) per coil. To simply levitate the handle the device draws only 100mA at 24V or 2.4W through the lower three lifting coils and almost no current in the upper coils.

While the PWM coil drivers are much more efficient than analog coil drivers (1.8W *vs* 16W), the current settling time is significantly higher (700μs *vs* 150μs). Current research addresses this issue.

Experiments carried out by the authors have shown that a periodic current change of ΔI=2mA at 1000 Hz, corresponding to an acceleration of approximately 0.1 m/s/s is about the smallest noticeable by users. Therefore the minimum required PWM resolution N_{PWM} in bits is:

$$N_{PWM} = \text{ceiling}[\log_2 \left(\frac{2V_{supply}}{R_{coil}\Delta I} \right)] \; . \qquad (1)$$

With V_{supply}=24 V, R_{coil}=8 Ω and ΔI=2 mA the required resolution is 12 bits. With a clock frequency of 48 MHz the PWM frequency is then 24kHz for Sign/Magnitude PWM Control and 12 kHz for Anti-Phase PWM Control. Both control modes are shown in Figure 3. Figure 4 shows experimentally measured coil currents averaged over one PWM signal period *vs* the command current. As a result of the

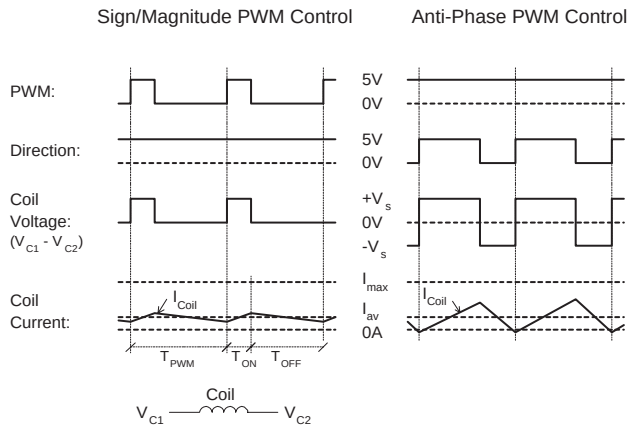

Figure 3. Coil driver PWM operation modes.

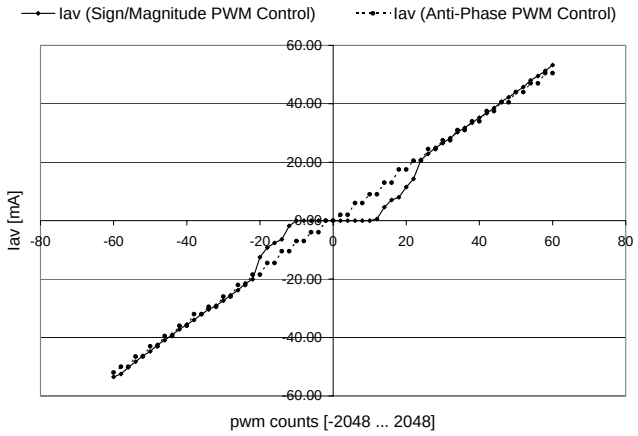

Figure 4. Measured averaged coil current *vs* command current. The command current is expressed in PWM counts ($0 \Leftrightarrow 0mA$, $\pm 2048 \Leftrightarrow \pm 3000mA$). (a) Sign/Magnitude and (b) Anti-Phase PWM Control operation.

minimum required on/off-time (specified as 1 μs) of the H-bridge LMD18200, curve (a) shows nonlinear behavior around the control point at zero. Given that 1 μs corresponds to 50 counts, one can see that the H-bridge works well down to 480 ns, exhibits a nonlinear behavior between 240-480 ns and shows a dead-zone for 0-240 ns. Since in general linearity at the control point is prefered one would want to choose the Anti-Phase PWM Control mode. This is not possible for the following two reasons: (1) With a 48 MHz system clock and a minimum PWM frequency of 24 kHz (above the human hearing range) the achievable resolution is only 11 bits, (2) A large current ripple ΔI_r introduces a significant disturbance on the handle. For example, with a current setpoint of 0 A, the maximum voltage is applied across the coil in one direction for half the time of the PWM cycle, and the maximum voltage is applied in the opposite direction for the remaining half-time of the PWM cycle, leading to the following current ripple ΔI_r:

$$\Delta I_r = I_{max} e^{-\frac{T_{PWM}}{2\tau}} \approx I_{max} \frac{T_{PWM}}{2\tau} = \frac{V_{supply}}{2L_{coil} f_{PWM}} \; . \qquad (2)$$

When V_{supply}=24 V, I_{max}=3 A, the current ripple is ΔI_r = 208 mA, where in (2) the coil inductance, resistance and time constant are L_{coil}=2.4 mH, R_{coil}=8 Ω, τ=300 μs, and f_{PWM}=1/T_{PWM}=24 kHz.

As a result, Sign/Magnitude PWM Control is the preferred mode of operation. For further improvement one could dither the commanded current since the PWM frequency (24 kHz) is much larger then the control loop frequency (1 kHz). This would help increase the output accu-

racy to as much as 14 bits, without driving the H-bridges beyond their minimum on/off time specification.

DEVICE PROGRAMMING AND HYBRID SYSTEM MODEL API

Software for the PowerMouse is developed using the GNU C/C++ language tool set and the RTEMS real-time kernel. Both of these packages are open source and can be freely distributed. A Java-based monitoring and debugging program has been developed to interface to the processor through the BDM port and allow to peek, poke and graph any memory location without altering the source code.

Currently the device operates as a stand-alone system allowing control sampling frequencies of 100-1000 Hz. The computation time of the complete 32-bit fixed point kinematics is 250 μs and can be substituted with a Jacobian-based kinematics requiring only 50 μs. A simple PID controller and the coil driving code is executed in just 50 μs. To achieve high position sensing accuracy, a significant part of the control period is spent waiting for optical sensor analog signals to settle. This is not idle processor time, as there are many communications and API tasks that are performed during this time.

The introduction of force feedback joysticks into the consumer market has been accompanied by the release of software application interfaces such as I-FORCE (see FEELit Overview, 1998) or DirectInput (see DirectX, 1999). Such software interfaces act like device drivers, providing programming models that are relatively independent of the underlying devices, and rely upon synthesizing complex behaviors from simple models such as springs, dampers, walls and barriers and other pre-defined effects. These behaviors are then downloaded into the haptic device where they are implemented by inexpensive "force processors" to avoid using the low bandwidth communications between the host and the haptic interface and to avoid host latency associated with common operating systems.

For complex virtual environment work, haptic worlds can be created using layered or more complex API's, such as the CyberImpact SDK (see CyberImpact SDK, 1997) or GHOST (see GHOST SDK, 1998). In GHOST, virtual objects are arranged in a spatial hierarchical structure, and the interaction between objects is computed and can be reflected to the haptic display automatically. The interaction dynamics and associated complexity require that the haptic device controller be a powerful processor.

Present haptic APIs use components distributed to the host and target processors. The amount of work allocated to the target is determined by its processing power and the bandwidth of communication with the host.

A novel approach to the division of computation between the host and the haptic device microcontroller has been developed by the authors and is summarized here. The model for the haptic interface processor is a finite state machine. Every state of this machine is characterized by a particular impedance controller. State transitions from one impedance controller to another occur whenever a set of linear inequality constraints, involving the mouse position, velocity and force, are satisfied.

Each state is characterized by a *state parameter* $P = [f_0, x_0, M, B, K]$ that defines the mass-spring-damper impedance emulated by the device:

$$f = f_0 + (Ms^2 + Bs + K)(x - x_0) \quad . \tag{3}$$

Here, f is a 6×1 vector of forces and torques exerted by the operator on the device, x is a 6×1 vector of the current mouse position and orientation (orientation is parameterized as the vector part of the Euler quaternion), x_0 is a 6×1 vector of the commanded mouse position and orientation, f_0 is a 6×1 vector of the commanded mouse force and torque, and M, B and K are 6×6 inertia, damping and stiffness parameters, respectively. If one considers the electrical network representation of the haptic interface, with forces corresponding to voltages and velocities to currents, the impedance (3) defines the Thevenin equivalent of the haptic device for small displacements around x_0.

A *condition* is a set $C = C(a_k, e_k,\ k \in I_C)$ of linear inequalities indexed by I_C:

$$a_k^T \begin{bmatrix} x \\ v \\ f \end{bmatrix} \leq e_k \quad , k \in I_C \quad , \tag{4}$$

where the 1×18 row vectors a_k^T and the scalars e_k, $k \in I_C$, are *condition parameters*.

A *state transition condition* occurs when all inequalities in a condition are satisfied.

A designated *exit state* is paired to each condition. Whenever a state transition condition is satisfied, a state transition to the exit state associated with the condition is executed.

The state parameters P_i, $i \in N_P$ and the condition parameters $[a_j^T, e_j]$, $j \in N_C$, where N_P and N_C are two index sets, are stored as arrays in haptic interface memory and are referenced by their array indices. The states $S_i = S_i(P_i, (C_{i_1}, S_{i_1}), (C_{i_2}, S_{i_2}), ..., (C_{i_{N_i}}, S_{i_{N_i}}))$ are stored as an array of indices and are referenced by their array index. The state indices point to the state parameter P_i and to N_i conditions with associated exit states (C_{i_1}, S_{i_1}), ... , $(C_{i_{N_i}}\ S_{i_{N_i}})$. The haptic interface exits the current state S_i

to enter a new state S_{i_k} if the k-th state transition condition occurs, *i.e.,* if all inequalities in the condition C_{i_k} are satisfied.

A discretized version of the control law that realizes the device impedance (3) is executed at a control sampling frequency f_s, with $f_s = 1000$ Hz in the current implementation of the PowerMouse API. During each sampling interval, the conditions are verified sequentially. When a state transition condition occurs, the designated exit state is entered. State transitions can also be triggered by external events such as the user pressing a button.

External inputs or current state positions, velocities or forces can be bound to the commanded force or position associated with a state.

The host API allows a programmer to load, copy, move, delete and read the states, the state parameters, and the state conditions and associated exit states. In general, host manipulation of these parameters cannot be accomplished during a single sampling time $1/f_s$. Assuming that each state has 20 inequalities of the type specified in (4), and that each of the entries of the matrices specified in (3) and (4) is stored in four bytes, roughly 10,000 states can be stored in the memory local to the haptic controller.

It is the responsibility of the host to ensure that all states, conditions, and transitions are properly set up with valid references and parameters. Since the state machine functionality of the microcontroller is initially disabled, it is possible for the host to initialize the system in any order. However, note that once the state machine is enabled, it is still possible to modify values but careful attention should be paid to the order of update.

Three examples of haptic effects and their implementation using hybrid dynamical systems are presented next. For simplicity, only scalar examples are considered.

Stiff wall: A unilateral or wall constraint as illustrated in Figure 5.A can be implemented using the finite-state machine of Figure 5.B. Zero stiffness (K=0) is applied during free motion and maximum stable device stiffness ($K = K_{max}$) is applied inside the wall. The transition state S_2, valid for a single sampling cycle, produces a braking pulse equivalent to high damping upon penetration. This has been shown to reduce the wall penetration and produce perceptually stiffer walls (Salcudean and Vlaar, 1997). The states are defined as follows:

$$S_1 = S_1([0, x_0, M, B_1, 0], \{-x \leq -x_0\}, S_2) \quad (5)$$
$$S_2 = S_2([0, x_0, M, B_2 + B_{brake}, 0], \{0 \cdot x \leq 0\}, S_3) \quad (6)$$
$$S_3 = S_3([0, x_0, M, B_2, K_{max}], \{x \leq x_0\}, S_1) \quad (7)$$

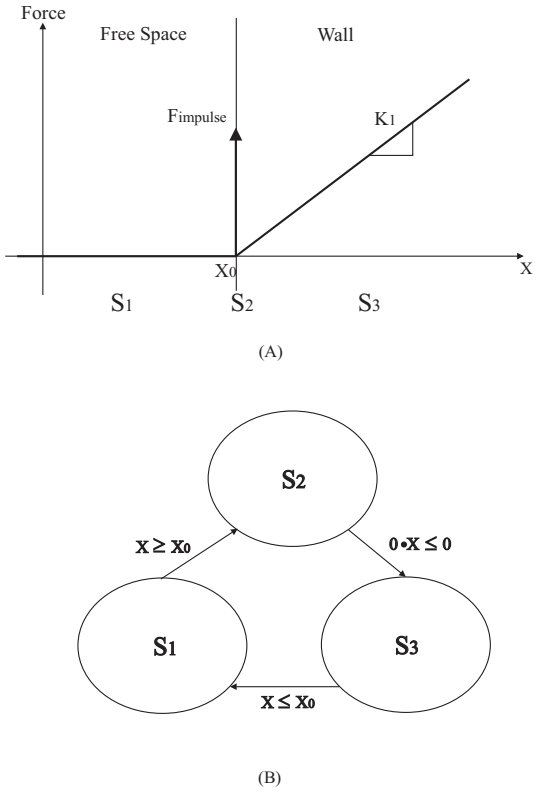

Figure 5. Implementation of a stiff wall with a hybrid system.

Button: As illustrated in Figure 6, a button can be implemented as a five-state machine with the following states

$$S_1 = S_1([0, 0, M_1, B_1, 0], \{-x \leq 0\}, S_2) \quad (8)$$
$$S_2 = S_2([0, 0, M_2, B_2, K_2], \quad (\{x \leq 0\}, S_1), (\{-x \leq -x_0\}, S_3)) \quad (9)$$
$$S_3 = S_3([F_{bounce}, x_0, M_3, B_3, 0], \quad (\{x \leq x_0\}, S_2), (\{-x \leq -x_1\}, S_4)) \quad (10)$$
$$S_4 = S_4([0, x_1, M_4, B_4, K_4], \quad (\{x \leq x_1\}, S_3), (\{-x \leq -x_2\}, S_5)) \quad (11)$$
$$S_5 = S_5([F_{sat}, x_2, M_5, B_5, 0], \{x \leq x_2\}, S_4) \ . \quad (12)$$

Stick-slip friction: As illustrated in Figure 7, a simple model of static friction can be implemented using a Karnopp model, as done in (Salcudean and Vlaar, 1997). The haptic device "sticks" when the velocity drops under a threshold, and starts slipping when the user force exceeds a threshold. The states are defined as follows:

$$S_1 = S_1([0, x(t - (1/f_s)), M, B_1, 0],$$

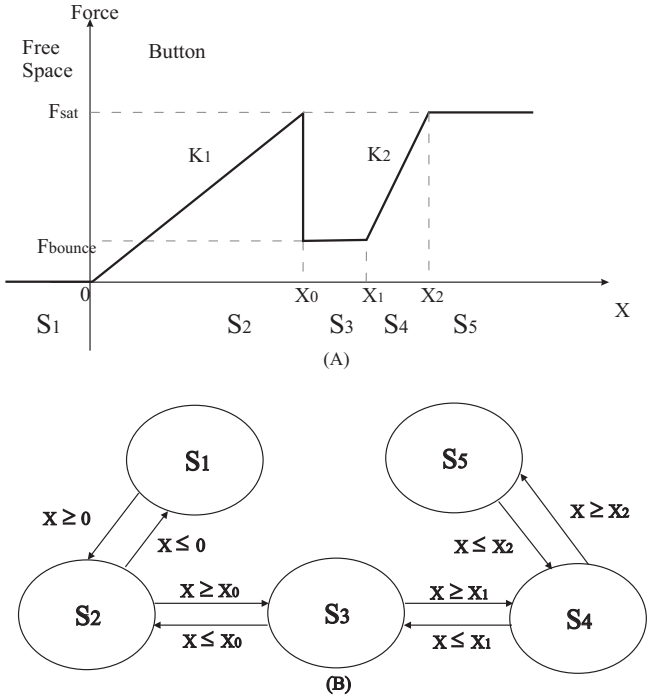

Figure 6. Implementation of a button with a hybrid system.

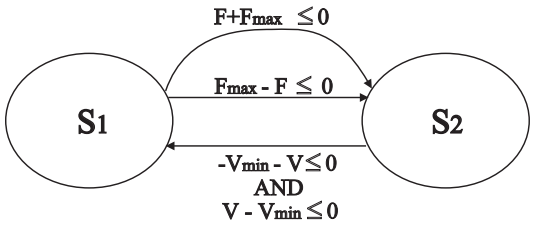

Figure 7. Implementation of stick-slip friction with a hybrid system.

$$(\{f \leq -f_{max}\}, S_2), (\{-f \leq -f_{max}\}, S_2)) \quad (13)$$

$$S_2 = S_2([0, x(t-(1/f_s)), M, B_2, K_2], \\ (\{v \leq v_{min}\ ,\ -v \leq v_{min}\}, S_1)) \quad (14)$$

The finite state machine approach to the design of the PowerMouse API has been motivated by the need to minimize communication latency between the haptic interface and the host. For a haptic interface to be effective, it needs to exhibit a broad range of impedances or Z-width (Colgate and Brown, 1994), but, as illustrated by the above examples, it also needs to switch from one impedance to another very quickly. The proposed API enables such switching with very little haptic processor computational overhead.

The structure of the state machine proposed by the authors is simple, yet it supports the description of complex geometric and dynamic objects. For example, point tracing of geometric objects can be done based on local hyperplane approximations, where a_k and e_k of (4) represent the surface normal and a distance to it, respectively. The inclusion of the state (position and velocity) in (4) also allows the implementation of sliding mode controllers.

The proposed API structure is memory intensive, but requires relatively little processor time as only array referencing and inner products are required. This matches not only the present PowerMouse controller board design, but also current hardware trends.

Generalizations of the proposed API model follow easily and will be implemented as necessary following further experimentation with the API. In particular, more complicated filters than the second-order mass-spring-damper could be postulated to act on both position and force in (3), and can comprise, for example, integral control on position or force error. As well, the state transition inequalities could be easily made non-linear to allow tracing of objects defined by implicit functions (Salisbury and Tarr, 1997).

The limited PowerMouse motion range requires an approach for gross positioning of the controlled virtual objects or teleoperation slaves. The simplest approach is indexing using velocity control and it involves defining a boundary layer in the haptic interface workspace. Whenever that boundary layer is entered, the mouse deflection is interpreted as a velocity command to the virtual object (Salcudean et al, 1995). A second approach involves using the device in rate control with force feedback while providing the user with a feel of the actual environment impedance (Zhu and Salcudean, 1995). Both methods have been implemented successfully in the context of teleoperation and are amenable to implementation using the hybrid dynamic system model of the device.

CONCLUSIONS

The electronic hardware design for the controller of a 6-DOF desk-top haptic interface has been summarized. Two modes of driving the actuator coils using pulse width modulation have been examined - Sign/Magnitude and Anti-Phase control. It was shown that in spite of the dead-band nonlinearity associated with Sign/Magnitude control, this method is preferable to Anti-Phase control because it displays a smaller current ripple, which translates into a smaller acceleration disturbance on the haptic interface.

A new approach to the design of haptic application programming interfaces has been presented. In this approach, the haptic interface emulates a finite state machine in which every state represents a particular impedance displayed to the user. Transitions from one state to another depend on the satisfaction of a set of linear inequality constraints. This approach allows the haptic interface to drastically change

impedances as required by a variety of haptic effects, without host intervention. Unlike approaches based on a collection of effects, the proposed API based on hybrid dynamic systems has a consistent formal structure that makes it easier to analyze and expand. Its formal structure and little computational overhead may allow the direct compilation of higher-level haptic models into its finite state machine components and is the topic of future research.

ACKNOWLEDGMENT

This work was supported by the IRIS/PRECARN Network of Centres of Excellence of Canada.

REFERENCES

Berkelman, P.J., Hollis, R.L. and Baraff, D., "Interaction with a Realtime Dynamic Environment Simulation using a Magnetic Levitation Haptic Interface Device." In *Proc. IEEE Intl. Conf. Robotics and Automation*, pp. 3261-3266, Detroit, May 1999.

Colgate, J.E. and Brown, J.M., "Factors affecting the *Z*-width of a haptic display." In *Proc. 1994 IEEE Int. Conf. on Robotics and Automation*, pp. 3205-3210, 1994.

Massie, T.M. and Salisbury, K., "The PHANToM Haptic Interface: a Device for Probing Virtual Objects", in *Proc. 3rd Symp. Haptic Interfaces for Virtual Environments and Teleoperator Systems*, DSC-1, pp. 295-301, Chicago, IL., Nov. 1994.

Salcudean, S.E., Wong, N.M., and Hollis R.L., "Design and Control of a Force-Reflecting Teleoperation System with Magnetically Levitated Master and Wrist". *IEEE Transactions on Robotics and Automation*, Vol. 11, No. 6, pp. 844-858, December 1995.

Salcudean, S.E. and Parker, N.R., "6-DOF Desk-Top Voice-Coil Joystick", *6th Annual Symposium on Haptic Interfaces for Virtual Environments and Teleoperation Systems, Intl. Mech. Eng. Congr. Exp., (ASME Winter Annual Meeting)*, DSC-Vol. 61, pp. 131–138, Dallas, Texas, November 16-21, 1997. Also, see http://www.ece.ubc.ca/~tims/maglev.html

Salcudean, S.E. and Vlaar, T. "On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input-Output Device", *ASME Journal of Dynamics, Measurement and Control*, 119:127–132, March 1997.

Salisbury, K., Brock, D., Massie, T., Swarup, N., Zilles, C., "Haptic Rendering: Programming Touch Interaction with Virtual Objects", in *Proc. 1995 Symposium on Interactive 3D Graphics*, Monterey, April, 1995, ACM, pp. 123-130.

Salisbury, K., and Tarr, C., "Haptic Rendering of Surfaces Defined by Implicit Functions", in *Proc. 6th Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, DSC-61, pp. 61-67, Dallas, TX, Nov. 1997.

Zhu, M. and Salcudean, S.E., "Achieving Transparency for Teleoperator Systems under Position and Rate Control", *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*, Vol. 2, pp. 7-12, August 5-9, 1995, Pittsburgh, PA.

CyberImpact Software Development Kit Ver. 2, Cybernet Systems Corporation, April 1997.

DirectX 6.1 SDK Documentation, Microsoft Corporation, 1999, http://www.microsoft.com/directx/pavilion/dinput

FEELit Overview and API Documentation, Rev. 0.97, http://www.force-feedback.com/iforce

GHOST SDK Programmer's Guide Version 2, SensAble Technologies Inc., July 1998, http://www.sensable.com/