Protecting the PHANToM from the Programmer

Rob Kooper, John Barrus, David Ratajczak, David Parsons

TR96-20 December 1996

Abstract

Anytime you connect motors and amplifiers to a computer, you make it possible for the computer to break something or hurt someone. However, careful design of hardware and software can permit full use of a device within limits imposed by safety concerns. It is trivial to cause one such device, the Phantom [Massie and Salisbury 94] from SensAble Technologies, to damage itself and in fact it is quite difficult not to do so. In this paper, we describe a robust solution to this problem.

Revision History

1. First version, TR96-20, October 2, 1996.

Introduction

Programming SensAble Technologies Phantom is difficult for many reasons, one of the foremost being that it is quite easy to write a program that will destroy the machine, as is shown in Example 1. This program will cause the Phantom arm to accelerate until it reaches its mechanical stops. High electrical currents will continue to flow to the motors and the insulation on the motor windings will melt and short out the motors.

```
#include <phantom.h>
void main(void) {
    /* Torque in Newtons */
    setPhantomTorques(10, 10, 10);
    sleep(1000);
}
```

Example 1. Application that will break the Phantom.

At MERL, we have been working on a system which makes it impossible for a programmer to hurt the Phantom in any way. This system is a combination of hardware and software which monitors both the Phantom and the actions of the software driving it, thus eliminating dangerous or fatal software commands.

The system protects the Phantom in four ways:

1. Thermal protection: If the motors are driven with too high of a current for too long, the amplifiers are disabled while the motors cool down.

2. Speed limits: If the end of the arm accelerates beyond a certain maximum speed, no additional torque is applied to the arm.

3. Torque limits: Torque requests sent to the Phantom motors are kept within safe limits.

4. Workspace limits: Resistive forces are applied in order to decelerate the Phantom when the user moves it too close to its mechanical workspace constraints.

Maximum speed, torque, and workspace limits are built into our safety system, but the programmer has the option of specifying more conservative limits while debugging Phantom programs.

In addition, the system provides some convenience to the application programmer in that it automatically provides compensation forces to counter unavoidable mechanical imbalance of the Phantom arm and also manages conversions which allow the programmer to work in meters in the Cartesian coordinate system instead of in the configuration space of the Phantom's arm.

Design Philosophy

Providing a safe environment for the Phantom requires careful consideration of the entire system. For instance, if some of the safety features are implemented in software, that software must not be influenced by factors which are out of the control of the architect of the system, such as bugs in other people’s programs. This consideration moved us to develop an embedded controller which removes the possibility of someone else’s software disabling our safeguards. An additional advantage of using an embedded controller is the opportunity to take over some of the processing normally done by the application computer, leaving more time for the application to run.

The following are some of the things we considered when designing the Phantom safety system:

- Never assume motors are at ambient temperature when application starts.
- Provide for absolute torque (current) limits at all times.
- Do as many calculations as possible on dedicated processor(s) to reduce the workload on the application computer.
- Reduce or eliminate complexity for the programmer.
- Allow safe addition and removal of objects while Phantom is operating.

Controller Attributes

Based on the above design philosophy, we created a controller for the Phantom which has the following attributes:

- Thermal simulation runs on embedded controller.
- Thermal simulation cannot be stopped by removing power or by programmer error and thermal protection cannot be shut off.
- Automatic conversion between Cartesian coordinate system and Phantom configuration space. Conversion from force vector to torques and from arm angles to XYZ position handled by controller in order to reduce computational load on application computer.
- Temperature, position, velocity, and error information provided to application programmer on request or at specified intervals.
- Compensation forces calculated to balance Phantom arm in all positions.
- Eliminates dangerous forces caused by new objects appearing in same location as users hand.

In order to accomplish the above tasks, we provide additional processors between the Phantom hardware and the application computer. One of those processors is essentially an embedded controller which drives the Phantom directly using information received from the application computer over an Ethernet connection (see Figure 1).

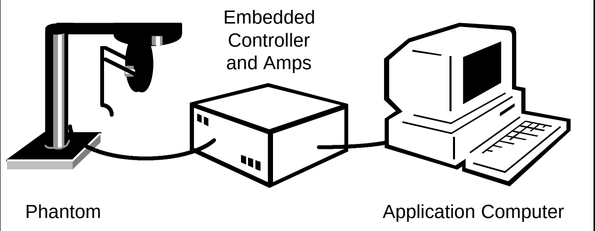

Figure 1 Phantom setup

Using this controller, the programmer no longer needs to set and read bits directly on the Phantom plug-in board. The user can create small packets of information, like force vector requests, and convey those directly to the controller over the Ethernet. Information is conveyed back to the application using the same communication channel. Since this is network based, many computers can receive the same information at the same time, although only one can have direct control of the Phantom.

Controller Implementation

A library of abstract C++ classes is provided to the application programmer. These classes provide network connectivity, temperature checks and predefined objects and are used to communicate with the controller. This enables the programmer to quickly build an application that can connect with the Phantom controller and issue commands and display Phantom specific information.

The current implementation of the controller is based on a 90Mhz Pentium running a real time UNIX variant (LynxOS). The communication between an application computer and the controller is handled by TCP/IP over Ethernet.

Thermal Protection

Since thermal protection is so vital, a unique solution was developed to prevent the motors from burning out. In addition to the thermal simulation of the motors running on the controller, we designed and built an electrical system which mimics the thermal performance of the motors. The voltage at one point in the electronic circuit precisely tracks the heating and cooling of the motors' internal windings and housing. Powering down the hardware has no affect on the cooling of the motors nor does it adversely affect the analog electronic simulation. The tracking voltage is always measured on start up to determine the current temperature of the motors for the software simulation. Additional electronics monitor the tracking voltage and automatically shut off the amplifiers if the voltages, and therefore the internal motor temperatures, reach dangerous levels. Even if the controller suffers an unexpected software failure, the motors will still be protected.

Torque and Velocity Limits

The Phantom controller converts the force vectors supplied by the user, or calculated internally, to torques. After this calculation, and before supplying these values to the Phantom, the controller checks to see if these values are within the range specified by the user. If the not, the Phantom controller scales the force vector to keep it within range.

When converting the angles of the Phantom arm to an XYZ position, the Phantom controller calculates the velocity of the Phantom. If the velocity is outside of the range supplied by the user, the Phantom controller will not apply a force until the velocity comes within the set limits.

Workspace Limits

One of the predefined objects in the controller is a model of the reachable workspace of the Phantom. This model is used to decelerate the Phantom whenever the probe approaches either the limits of the Phantom's reach or fixed obstacles in the workspace, such as a desktop. This provides an important operating safety feature, and also supplies useful haptic feedback to the user. The geometry and haptic rendering algorithms of the workspace model can be easily modified.

SensAble’s Approach

SensAble Technologies has a Phantom library (called GHOST) which does all of the thermal calculations and works with a watchdog timer on the amplifier box to reduce the chance of burning out motors. However, it has no provisions for the situation in which a person using the Phantom might shut off the software when the motors are hot and then restart the program immediately. On restart, the GHOST library must assume that the motors are at ambient temperature because the GHOST library has no way of measuring the real temperature. If the motors are hot on startup then it is possible to burn out the motors using SensAble’s software. In order to guarantee that the motors are at ambient temperature, more than 15 minutes would have to pass before restarting the program because of the amount of time it takes for the motor housing to cool.

Future Work

Unfortunately, due to limitations in capacitor technology, the electronics that mimic the motor temperature are not very reliable or stable and require frequent recalibration. Other methods of thermal tracking are being pursued, to augment the software based thermal simulation.

The speed of the Ethernet communication is too slow. Maximum round-trip communication using the current embedded controller is about 500 Hz. This low update rate introduces instabilities when representing stiff objects like walls. Although rewriting the Ethernet drivers for our system might speed it up a bit, we are also considering other communication technologies like FireWire, Univeral Serial Bus and High Performance Parallel Interface when they become more widely available. Also, the use of a simple stiff wall model internally in the controller like the one used in the UNC Phantom library might be an effective alternative [Mark et al. '96].

Another way to alleviate problems with communication speed is to remove the need for high speed communication completely. If a haptic model could be loaded into the controller and the controller could then do all of the necessary calculations for delivering the correct force to the user, the applications computer would not have to worry about haptic calculations at all and the Ethernet communications latency would no longer be a problem. In the near future we will be considering a model format or file format for storing and communicating haptic information to the controller. This format will provide for transmission of not only haptic object geometry but also surface properties such as friction. The controller could implement a friction model such as the one described in [Zilles and Salisbury '95].

ACKNOWLEDGEMENTS

The authors would like to thank the employees of SensAble Technologies for the quick response and support every time we broke our Phantom.

References

[Mark et al. '96] William R. Mark, Scott C. Randolph, Mark Finch, James M. Van Verth, and Russell M. Taylor II, "*Adding Force Feedback to Graphics Systems: Issues and Solutions*", Computer Graphics (SIGGRAPH'96 Proceedings), 30(2), pp. 447-452.

[Massie and Salisbury '94] T. Massie and K. Salisbury, "*The PHANToM Haptic Interface: A Device for Probing Virtual Objects*", Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems, Chicago, IL, November 1994.

[Zilles and Salisbury '95] Craig B. Zilles and J. Kenneth Salisbury, "*A Constraint-based God-object Method For Haptic Display*", Proceedings of the International Conference on Intelligent Robots and Systems (IROS '95).