RL-TR-94-79
Final Technical Report
June 1994

EXPLOITATION OF VIRTUAL REALITY ARCHITECTURES

The MITRE Corporation

H. Veron, D.A. Southard, R.B. Mitchell, P.J. Hezel,
J.L. Segal, and H.C. Masterman

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-79 has been reviewed and is approved for publication.

APPROVED: Robert M. Flo

ROBERT M. FLO
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1994	Final --------

4. TITLE AND SUBTITLE
EXPLOITATION OF VIRTUAL REALITY ARCHITECTURES

5. FUNDING NUMBERS
C - F19628-94-C-0001
PE - 62702F
PR - MOIE
TA - 74
WU - 05

6. AUTHOR(S)
H. Veron, D.A. Southard, R.B. Mitchell, P.J. Hezel, J.L. Segal, and H.C. Masterman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The MITRE Corporation
Center for Air Force C3 Systems
202 Burlington Rd
Bedford MA 01730-1420

8. PERFORMING ORGANIZATION REPORT NUMBER
MTR 93B0000191

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-79

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Robert M. Flo/C3AB/(315) 330-2805

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A computer interface technology known as virtual reality (VR) has received a great deal of attention because it provides a new paradigm for interfacing humans to computer-generated synthetic environments. VR is a natural outgrowth of several enabling technologies: workstations, computer graphics, visualization, displays, interactive devices, and software. The user can then directly manipulate simulated objects and leave behind the limitations of the physical world. The goal of this project was to examine VR for potential application in command, control, battle management, and situation awareness. Work focused on establishing and developing an optimized hardware and software architecture for a variety of applications. The purpose of this report is to provide a technical summary of MITRE's findings. The report begins with a user-interface device description and evaluation and a description of the software requirements for integrating these devices in a VR platform and the implications on the software architecture. The report then describes the Virtual Environment Architecture (VEA), the software architecture used to develop a real-time stand-alone interactive VR platform, including rationale, tradeoffs, performance issues, and use of a knowledge-base. Finally, the report describes a scenario developed for demonstrating the utility of the VR platform in a battle management prototype application.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Virtual Reality, Display Technology, Human-Computer Interface	134
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ABSTRACT

We evaluated the emerging technology termed "virtual reality" as a potential technique for use in command, control, battle management, and situation awareness applications. In particular, we assessed several novel user-interface devices, which constitute a representative range of technologies and vendors for position trackers, 3D controllers, whole-hand input "gloves" and head-mounted displays. We analyzed the software requirements for virtual environments, and compared several software architectures published in the literature. We developed a virtual environment architecture, which emphasizes an event-driven, multi-processing approach. As a demonstration prototype, we incorporated a knowledge-based "battle manager's assistant" into a virtual environment. Our investigation revealed that environment modeling and interactive response are critical performance factors. Our virtual environment architecture addresses these performance issues, and provides an extensible infrastructure that supports new devices and applications.

EXECUTIVE SUMMARY

A computer interface technology known as virtual reality (VR) has received a great deal of public attention, because it provides a new paradigm for interfacing humans to computer-generated synthetic environments. VR is a natural outgrowth of several enabling technologies: workstations, computer graphics, visualization, displays, interactive devices, and software. VR immerses the user in a simulated computer environment. The user can then directly manipulate simulated objects, and leave behind limitations of the physical world.

GOAL

The goal of this project was to examine VR for possible application in command, control, battle management, and situation awareness. We focused our attention on establishing and developing an optimized hardware and software architecture for a variety of applications.

ACCOMPLISHMENTS

We made advances in the following areas:

- *Device assessment.* We tested position trackers in detail. Electro-magnetic trackers showed the best stability and range, but suffer from magnetic interference from ferrous and nonferrous metals in the environment. Optical and acoustic trackers are immune to metals, but have stringent line-of-sight requirements. Both magnetic and acoustic technologies are currently encumbered by tethers. Optical sensors usually have no tethers; however, the model we tested supported only position tracking, with no orientation information.

- *Head-mounted displays.* A head-mounted display (HMD) is an especially important component in a VR system, because it provides visual immersion in the virtual environment. All current HMDs are severely limited, either in color, resolution, weight, or mobility (by the presence of tethers). However, HMD technology is constantly improving. We recommend a technically achievable set of goals for future HMD designs. For the time being, large screen stereoscopic projection displays offer a viable alternative to HMDs, for group viewing applications.

- *Software architecture.* A significant result of our work is a generic architecture for virtual environment applications, termed the virtual environment architecture (VEA). We surveyed several VR software architectures, and analyzed their requirements. VEA offers several unique solutions to recurring design themes:

- *Device abstraction* is implemented by organizing user-interface devices by *mode* of interaction. Interactive modes include: button, gesture, graphic, locator, posture, sound, text, speech, valuator. The interactive modes have a close correspondence to human senses and abilities: tactile (button, valuator), visual (graphic), aural (sound), haptic (locator, posture), speech (text). Device independence is encouraged by layering the device interfaces. A *filter* layer converts device-dependent information into device-independent interactive modalities. A *manager* layer interprets modal interactions, converting them into task-specific actions.

- *Parallelism* is supported using shared-memory multiprocessing. This type of architecture is best suited to today's multi-CPU workstations.

- *Object-oriented modeling and simulation* is supported by an event-driven "software bus" architecture. Events are moderated by a real-time clock, which can be controlled to run at faster or slower than real-time.

- *Rapid prototyping* is supported by a flexible configuration file, which can be used to set-up new applications from existing modules, without having to modify or recompile program code.

- *Prototype demonstration.* We built a prototype demonstration using VEA. This application used a knowledge-base tool, NASA's C-Language Integrated Production System (CLIPS), to construct a "battle managers assistant." The demonstration depicted an operational area, populated with enemy and friendly vehicles and facilities. The user can query the virtual environment using voice commands. The knowledge base responds by identifying potential threats, by showing context-sensitive information, and by recommending assets appropriate to counter those threats. The knowledge base also models object behaviors, so that simulated objects can respond to commands appropriately.

The demonstration was implemented on a 4-CPU Silicon Graphics 4D/340 RealityEngine workstation. The user interacts with the demonstration using voice or typed commands, a 3D "wand" for pointing and picking objects, and our large screen stereoscopic display. Results indicate that VEA effectively distributed the simulation and graphics computing load over multiple processors, providing an improvement in graphics throughput and fidelity, compared with a single-threaded approach.

IMPACT

This work has resulted in several technical papers and presentations, which have placed MITRE in a position as a recognized contributor to VR technology. The VR platform built for this project is the corporate vehicle for demonstrating the capabilities of VR.

Future work in this area will focus on integrating more advanced interactive techniques, and improving interactive response times. We intend to extend this architecture to a multi-platform, multiple-user, distributed virtual environment capability.

ACKNOWLEDGMENTS

The authors wish to acknowledge Air Force Materiel Command's Rome Laboratory, Griffiss Air Force Base, for supporting the work reported in this document.

TABLE OF CONTENTS

SECTION PAGE

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

Until recently, interacting with a computer has been confined to a keyboard, a mouse and a two-dimensional (2D) display screen. With the evolution of interactive three-dimensional (3D) computer graphics, new devices have been developed that provide new ways to interact with the displayed computer generated environment. In their most basic form, these devices provide for the input of 3D position coordinates. Other devices add 3D orientation data as well. Devices have also been developed that utilize more human senses. These include tactile feedback, force feedback, aural and stereoscopic displays. Some devices also monitor the user's hand and finger positions, providing the opportunity for whole-hand input [Sturman 92]. With whole-hand input, the user can grab and interact in a natural way with a computer generated display.

Display technology has progressed to the point where the user can begin to have a true sense of presence in a computer-generated environment. The combination of real-time 3D computer graphics with shading and texture mapping, high resolution stereoscopic large screen or head-mounted displays, along with these novel user interface devices, has been termed *virtual reality* (VR). When integrated with appropriate software, VR creates a new and exciting paradigm for human-computer interaction. It is potentially an immersive, interactive, and intuitive interface to a computer generated environment.

For the military, VR offers unique opportunities in the areas of simulation, training and situation assessment. VR allows the development of a potential battle environment, including terrain profile, allocation of resources, position and disposition of the threat. Using a knowledge base, the commander and his staff can view the implications of potential strategies and tactics, adapt them for the situation and select the most appropriate one for the course of action.

1.1 SCOPE

The purpose of this program was to investigate VR for battle management applications. This was accomplished by an evaluation of the performance characteristics of several new VR devices available on the market. This evaluation was further aided by the design and development of a software architecture that integrated selected VR devices for use in a single-user VR platform. The utility of such a platform was demonstrated by applying it to a prototype battle management application.

1.2 ORGANIZATION

This report is structured as follows:

Section 2 contains the detailed user interface device description and evaluation.

Section 3 describes software requirements for integrating these devices in a VR platform and implications on the software architecture. Selected architectures are used to illustrate how these requirements are implemented. The MITRE software architecture is introduced and compared to these approaches.

Section 4 describes Virtual Environment Architecture (VEA), the unique software architecture used to develop a real time stand-alone interactive VR platform. We also discuss rationale, tradeoffs and performance.

Section 5 describes a unique use of a knowledge base in our VR architecture.

Section 6 describes a scenario developed for demonstrating the utility of the VR platform in a battle management prototype application.

Section 7 provides a summary of our findings and concludes this report.

SECTION 2

USER INTERFACE DEVICES

VR requires the user to operate in 3D space using sight, sound, and body movements, involving head, hands, fingers, as well as sensations of touch. In theory, even the senses of smell and taste could be included, but little has been reported to date in these areas. Input devices can be categorized according to the operative human sense, and by the number of degrees of freedom offered. Each degree of freedom can be parsed, processed and analyzed within the host computer for a specific application.

The Virtual Environment Architecture (VEA) developed at MITRE, and described in Section 4, groups devices into a set of categories that are common to a wide range of input devices. This facilitates the device interface to the software because the data is interpreted independently of the specific device that generated it. For example devices that generate *x*, *y*, *z*, *pitch*, *yaw* and *roll* data are grouped as "Locators." Hand positions, specified by finger angles, are grouped as "Postures." Switch settings are termed "Buttons." Devices are thus categorized by functionality. Typical device categories are summarized in table 1.

Table 1. Device Categories

Category	*Input/Output*	*Examples*
Button	Input	mouse, button box
Locator	Input	trackers, 3D joystick
Posture	Input	hand, arms, body
Pointer	Input	mouse, trackball
Valuator	Input	dials
Sound	Input/Output	buzzer, MIDI
Text	Input/Output	speech, keyboard, alpha display
Graphic	Output	3D display

Physical interface devices may include multiple categories. For example, some products instrument hand and finger motion. The Exos Dexterous HandMaster (DHM) uses an electro-magnetic sensor to locate the hand in space and Hall-effect sensors to measure finger angles. Therefore, glove type input devices are Locators, due to the position sensor, plus Postures, because of the finger sensors. A full body suit could be characterized as a Posture and Locator. Many viable combinations are possible. As new types of input devices are characterized, we would expect new classes to be added.

2.1 DEVICE EVALUATION OVERVIEW

Because VR devices often perform different functions, comparisons among them are difficult. The approach used in this report will be to compare the devices that perform similar functions, and then to make recommendations for the integrated use of these devices. A discussion of the visual displays will be covered in Section 2.6. The groups into which we divide current VR interface devices are:

- **Position Sensors**
  - *Absolute position trackers:* determine the 3D position and orientation of a physical object that is coupled with an object in the virtual space.
  - *Relative position controllers:* "fly-through controllers" control the movement through a virtual space by means of a stationary device that does not require physical movement of the user through real space.
- **Gloves** — enable the computer to monitor the motions of the user's hand for whole-hand input.
- **Voice** — input words and phrases with speech (hands-off command input).
- **Feedback** — acoustical and tactile output in conjunction with the visual output add additional realism and flexibility.
- **Display** — visual presentation of the (dynamic) virtual environment.

2.2 ABSOLUTE POSITION TRACKERS

The idea behind the absolute position tracker is that positions in 3D physical space can be replicated in the virtual space. For example, if users move their hands 12 inches, they would like to see the image of their hands move the same 12 inches with the same orientation in the virtual space. Ideally, the user would like to have a position sensor that maps the real space correctly, with no error or distortion, so that the spatial relationships of real world objects in the virtual space can be presented accurately.

To evaluate the ability of position trackers to provide correct measurements, the users must establish required operating characteristics. In light of these requirements, the candidate devices must be determined, and a series of measurements established to determine how they achieve each of the required conditions. In addition, tradeoff analyses are performed and conditions that limit optimal tracker performance are determined.

2.2.1 Ideal Position Sensor

An ideal position sensor would provide:

- high rates of data acquisition to eliminate lag time
- high resolution and accuracy
- stable, repeatable measurements
- large spatial range of operation
- no interference from operating environment
- capability of multiple position sensors to operate concurrently

These ideal operating characteristics, however, are restricted by technological and physical limitations of the tracking system.

2.2.2 Evaluation Criteria

We evaluated the performance of several devices in both static and dynamic situations. We will offer a quantitative look at the performance of the sensors wherever possible, and a qualitative evaluation where necessary, according to the parameters below.

The position sensors are evaluated according to the following outline:

- Hardware configuration
  - type of position sensor
  - transmitter - receiver
  - mobility
  - tethers
- Manufacturer specifications of operation
  - degrees of freedom
  - reported range
  - resolution and accuracy
  - responsiveness: sample rate, data rate, update rate, and latency
  - maximum number of concurrent receivers

- Measured characteristics

  - range
  - repeatability
  - registration: correspondence of physical to virtual measurement
  - angle calibration
  - environmental interference
  - jitter

We assume that a high level of accuracy is necessary, even though in many applications we recognize that rather large inaccuracies in placement of virtual objects can be overcome by hand-eye coordination in the visual representation. For example, the virtual hand may be rotated to a different orientation than the real hand. Most of the time, the user can compensate by adjusting the position of his hand. Because many errors can be overcome in this way, it is difficult to define a range of acceptable errors. Since the need for accuracy depends upon the particular application, we state the quantitative results, from which judgments can be made about suitability for a particular application.

2.2.2.1 Hardware Configuration

This report describes:

- the type of each device and how it works
- the hardware that is associated with it
- and its physical arrangement in the workspace.

Most of the sensors have a transmitter that radiates some form of energy, and a receiver that detects the signal and calculates a position. Usually there are tethers required for transmitting data from the receiver to the controller electronics. The mobility of the position sensor will be described in terms of its ease of use within the workspace, based on the physical characteristics and the tethers.

2.2.2.2 Manufacturer specifications of operation

The *degrees of freedom* refers to the number of measurements that the device takes in order to specify position and orientation. Most devices measure six degrees of freedom to specify the position and orientation: three degrees of freedom for position, *x*, *y*, and *z*, and three degrees of freedom for orientation, *pitch*, *yaw*, and *roll*. However, some devices measure only position.

The *resolution* is the smallest change the sensor can detect. *Accuracy* is the range within which a given measurement is correct. Based on our experiments, the absolute accuracy of a measurement is significantly different from the specified accuracy.

The *responsiveness* is the rate at which the tracker can measure and transmit the position data to the host computer. It can be characterized by several factors: the *sampling rate* of the tracker, the *data rate*, the *update rate*, and the *latency*. The sample rate is a device parameter and is the rate at which the position of the sensor is monitored by the system. A high sample rate is good, as long as there are new position measurements. However, the possibility exists that measurements can be duplicated. The update rate is the number of computed positions per second. The data rate is the rate at which measurements can be transmitted to the computer. The best way of describing the responsiveness of the tracking system is by the *latency*, also referred to as *lag time*. Latency represents the time needed for the host to ask for the measurement and then receive it. It depends on the baud rate of the communication interface, the number of bytes transmitted, and on the sampling rate and data rate of the system. A small latency is most desirable because it allows the possibility of rendering graphics in real time. There is no universally accepted standard for latency, but studies have speculated that latency greater than 120 milliseconds impairs adaptation of the user [Meyer 92].

[Meyer 92] defines lag or latency as the elapsed time from the occurrence of an event until the event is observable by the host system. For the purposes of this report, latency is the sum of the following:

1. An average of one half of the sampling rate
2. The time for the host to request data from the device in question
3. The time for the device to transmit its data to the host
4. The time for the host to process the data and ultimately display the new result.

Items 2 and 4 are small compared with 1 and 3. Therefore, we estimated item 2 plus item 4 to be nominally 1 ms. If items 1 and 3 could each be reduced by an order of magnitude, we would have to measure items 2 and 4 with greater accuracy, rather than using a nominal value.

Example: A device supplies 24 serial bytes at 38.4 Kbps with a maximum update rate of 60 readings per second.

Latency = ((24 bytes @ 10 bits per byte)/38.4 Kbps) +
(1000ms per sec/(2.0 * 60 per sec)) + 1ms = 15.6 ms.

VR applications often require that multiple objects be tracked. The *maximum number of concurrent receivers* depends upon the tracking system architecture as well as on the type of

interference that can affect the ability to take measurements. It should be noted, however, that introducing multiple concurrent sensors may seriously affect other performance issues, including, but not limited to, lag time and update rate. This will be discussed as appropriate.

2.2.2.3 Measured Characteristics

Although the measurements we took to characterize each tracking system are in fact interrelated, we looked at each characteristic independently. For example, when looking at the range of the operability, it is difficult to maintain the same orientation, and therefore the changes in orientation that occur in the measurements as human error are indistinguishable from those that are tracking errors. Thus the measurements that we obtained are often over a sample area, and the results may be extended to the entire range of operation based on qualitative analysis of these and other results.

Range. The *range of operation* is difficult to characterize, since there are up to six numbers that are specified in each position measurement and an error in any one of those numbers may make a given measurement unreliable. A general range was first determined from the manufacturer specifications. Then we tried to determine first, how well the tracker performed within that range and second, where certain limitations occurred, based on the other criteria. For example, there is the absolute limit of operability of the device, dependent on transmitter ranges. Then there are fringe areas where the device may still be operable but not reliable. It is important also to look at a graphical image whose position is based on the tracker, to get a sense of what happens within the range of the tracker.

Repeatability. It is necessary that the position tracker give repeatable measurements. If users move an object in a virtual space with their hand, they want to be sure that the hand is in the same location when they try to move that object again. Large disparity will be intolerable. A look at the repeatability also gives insight into the range of operability.

Registration. The correspondence of the physical world measurements and virtual world measurements is not required for all VR applications. However, this will be necessary in applications where there is an overlay of the virtual environment on the physical environment. In such a case, it is important that the physical and virtual world positions correspond. When multiple sensors are used to track related objects, as in the case of using a head tracker with a head-mounted display (HMD) and gloves, good registration becomes a factor of prime importance. [Bryson 92-1].

Angle Calibration. The angles of the orientation should correspond to that which is expected. Each angle (*pitch*, *yaw*, *roll*) is examined independently. The specific angles relating the orientation of the sensor are ordered due to the graphics software associated with the specific device. That is, the rotations must be performed in a specified order about each axis. In some cases, the orientation angles become undefined at or near the extreme of the range of angles, thus resulting in erroneous measurements.

Environmental Interference. Interference with measurements by remote objects in the operating environment introduces significant errors in the measurements. Possibilities include reflection, noise, electromagnetic effects of wires and metals, and occlusion of the sensor or transmitter. These impose limitations on the type of operating environment.

Jitter. One of the other problems is the stability of the reported data from a position sensor at a selected fixed location. Jitter will often occur, which produces problems with the graphic image. In applications using a head tracker, this has the potential for causing problems with motion sickness. In any case the jitter needs to be minimized. This report qualitatively discusses the jitter observed with the devices.

Appendix A contains detailed measurements and evaluations of absolute position trackers. Table 2 summarizes their performance with appropriate comments regarding the measurements.

Table 2. Absolute Position Trackers

Vendor/ Model	Sensor Tech	Degrees of Freedom	Tethers	Range	Resolution	Latency (millisec)	Remarks
Logitech 3D/6D mouse	Acoustic	6	yes	mousemode: 2 ft. cube with 8 in. fringe head tracker mode: 7 ft. cube	.005 in. .02 in. in fringe 0.1 deg.	19.4	•require maintain line-of-sight between emitter and receiver •no apparent interference from reflections of acoustic energy off objects in environment •resolution, range, accuracy, and repeatability good up to specified area
Origin Instruments DynaSight	Optical/IR	3	no	4.9 ft. in 55 deg. cone	0.004 in. cross range 0.16 in. downrange	21.9	•require maintain line-of-sight between emitter and reflector (target) •reflections from surfaces other than target are large enough to lose track •excellent repeatability; other positional specifications are good within specified emitter angle •use of large targets provide greater range in exchange for resolution
Ascension Technology Flock of Birds	Magnetic (pulsed)	6	yes	±8 ft	.03 in 0.1 deg	12.3 (one sensor)	•environmental interference affects performance •can support 30 simultaneous receivers •one source, many receivers •single serial interface for all receivers
Polhemus 3Space Isotrack	Magnetic	6	yes	±2.5 ft	.03 in 0.1 deg	30.7 (one sensor)	•environmental interference affects performance •3Space Isotrak configuration tracked three receivers simultaneously •one source per receiver •each receiver has serial interface

*See section 2.2.2 for definition of latency

2.3 RELATIVE POSITION CONTROLLERS

Two commercially available relative trackers were evaluated, and they are discussed in this section.

2.3.1 Global 3D

The "GLOBAL 3D Controller," shown in figure 2.1, is manufactured by GLOBAL Devices of Granite Bay, CA. The 3D Controller comprises a sensing ball mounted on a base. To use this device, one holds the sensing ball and pushes, pulls, twists or turns in the direction desired. The 3D Controller is unique in that it provides tactile feedback to the user.

The 3D Controller has six switches which provide one bit of data indicating whether the user is pushing the device forward or backward, left or right, up or down. In addition, it has another set of six switches that indicate if the device is being rolled clockwise or counterclockwise, pitched up or down, or yawed right or left. Furthermore, the ball that the user is holding can vibrate at 15 different amplitudes and for 16 time intervals under software control. There is no inherent coupling among the various 3D Controller functions, and all must be controlled with user applications. No formal measurements were made to evaluate the Global 3D controller at this time.

2.3.2 SpaceBall

The "SpaceBall," shown in figure 2.2, is manufactured by SpaceBall, Incorporated of Billerica, MA. It comprises a sensing ball mounted on a base. In addition, it includes a pick switch right on the sensing ball, and eight function keys.

To use the SpaceBall, one holds the sensing ball and pushes in the x, y or z directions for translation, or twists in the *pitch*, *yaw* or *roll* directions for rotation. There is no inherent coupling among the various SpaceBall functions, and all must be defined with user applications. No formal measurements were made to evaluate the SpaceBall at this time.

2.4 GLOVES

We purchased the two types of gloves that are commonly used in VR research and development.

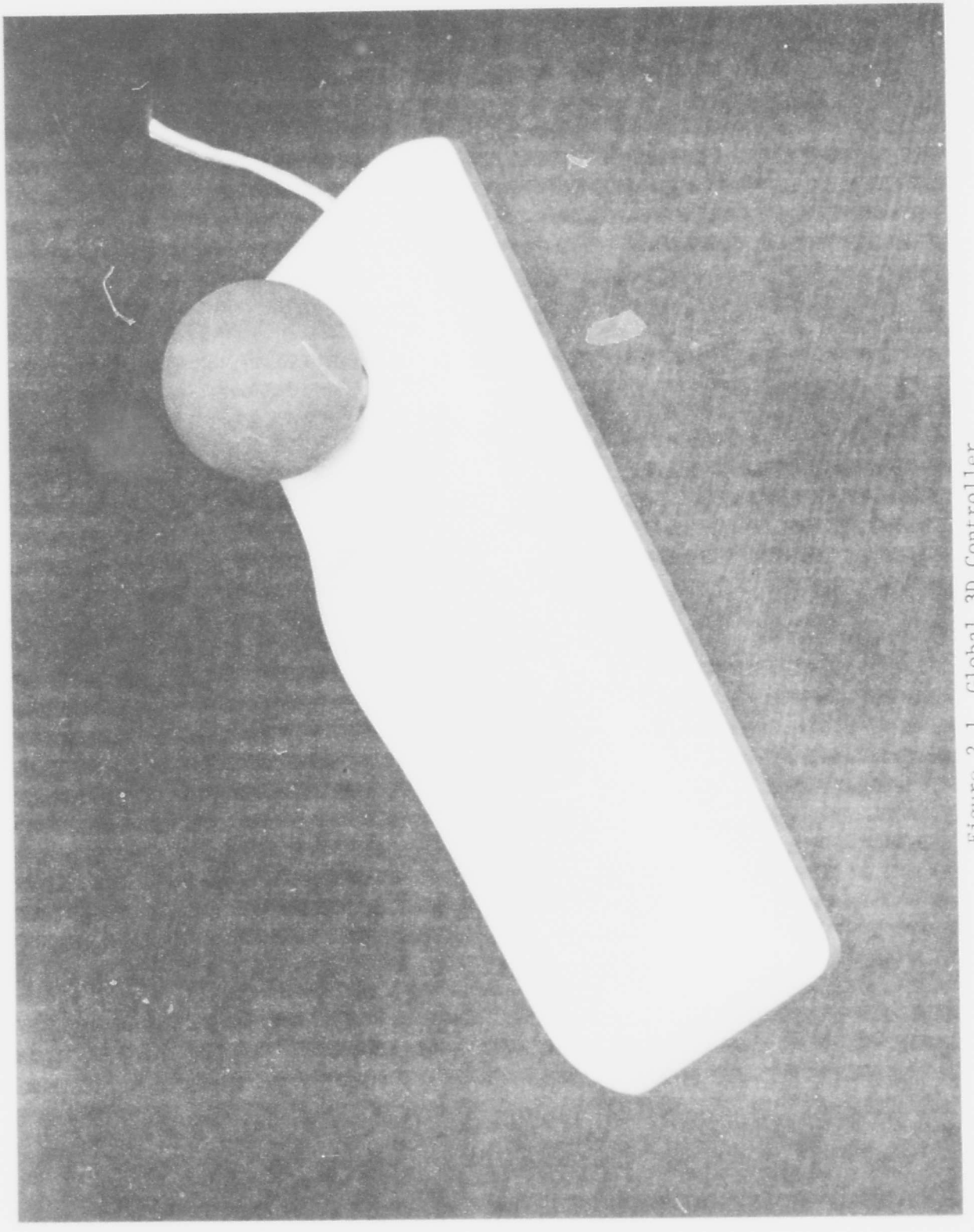

Figure 2.1 Global 3D Controller

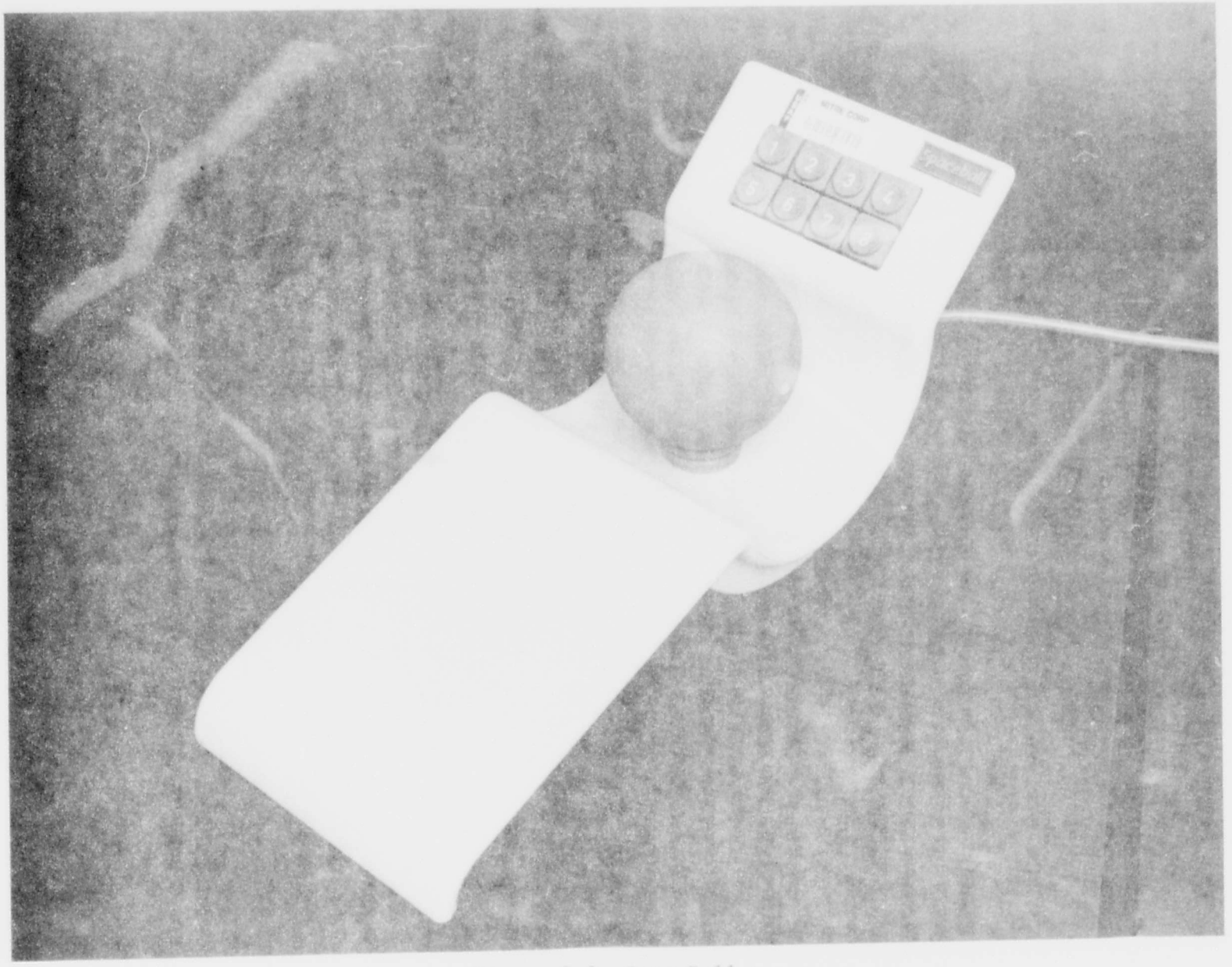

Figure 2.2 SpaceBall

2.4.1 Exos Dexterous HandMaster

The "Dexterous HandMaster" (DHM), shown in figure 2.3, is manufactured by Exos Corporation, of Woburn, MA. When placed on the user's hand, the DHM measures the location of the user's hand in *x, y, z, pitch, yaw, roll* and nine finger angles, combinations of which provide Locator and Posture events. Six angles are associated with the bending of the joints of the thumb, forefinger and middle finger. The other three angles measure the lateral motion of the fingers.

The *x, y, z, pitch, yaw* and *roll* data are obtained using a product made by Ascension Technology of Burlington, VT. The product is called "Flock of Birds" which is described in appendix A.3.

The DHM uses Hall-effect sensors to determine the finger angles. The sensors are controlled and measured by a control unit which provides the finger angles through a VME Bus interface. Despite numerous attempts to attach the DHM in a stable manner, crosstalk in the DHM was caused by almost any motion of the user's hand. While the theoretical resolution of the Hall-effect sensors, according to vendor specification, is about 0.1 degrees, with crosstalk, ± 10 degrees would be optimistic.

In addition the DHM is equipped with an EXOS TouchMaster™ which provides vibrotactile feedback to each finger under software control. The TouchMaster is interfaced by the VME Bus. The TouchMaster is still experimental and it was delivered with no tested application software. The effect of the TouchMaster is very subjective. Most users agreed that the vibrotactile feedback added something useful. However, many noted that it was sometimes difficult to feel which finger was stimulated. In addition, the TouchMaster is at an early stage of development and it is therefore cumbersome to wear. Each finger stimulator in the TouchMaster is held to a finger by a Velcro band. While most agree that tactile feedback is important to VR, most will also agree that a significant amount of research is required both from a hardware/software point of view and from a human factors/physiology point of view, to optimize the user interface.

The DHM is cumbersome both to put on and to wear. It is held on by seven Velcro bands and two elastic bands. After wearing it for as little as 15 minutes, some users reported tingling at the extremities of one or more finger tips. Due to symmetries in construction of the DHM, it is possible to put the thumb on backwards resulting in mirror-imaged readings.

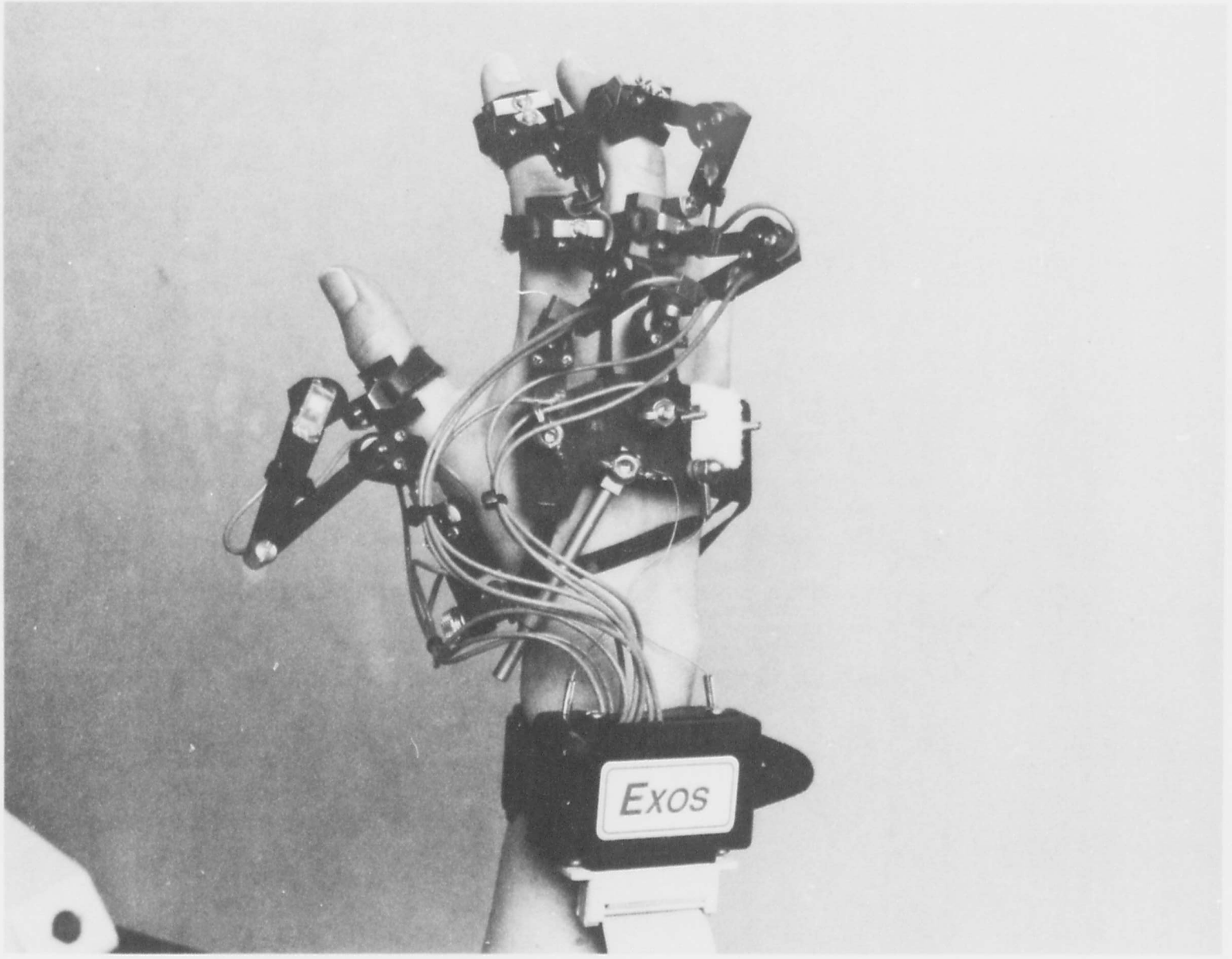

Figure 2.3 Exos Dexterous HandMaster (DHM)

2.4.2 VPL DataGlove

The VPL "DataGlove," shown in figure 2.4, is manufactured by VPL Corporation of Fremont, CA. When placed on the user's hand, the DataGlove provides a serial stream of data to the host computer. The data includes the location of the user's hand in *x, y, z, pitch, yaw, roll* and ten angles associated with the bending of the joints of the fingers.

The *x, y, z, pitch, yaw* and *roll* data are obtained using a 3Space Isotrack Sensor manufactured by Polhemus Corporation of Colchester, VT. The Isotrack Sensor includes a magnetic sensor which is attached to the back of the glove. A transmitter sets up a magnetic field that works with the sensor. The control box communicates with the host computer via an RS-232 port. The details of the operation of the Isotrack are contained in Appendix A.4.

The DataGlove has optical fibers sewn to the fingers. The more a joint is bent, the less light is transmitted through the fiber to the receiver. The light intensity received is inversely proportional to the magnitude of the finger angle.

The way the DataGlove is designed, crosstalk among the fingers is a major problem. In other words, bending one finger causes false readings on other fingers. This is due to the stretching of the Velcro glove as the fingers are flexed. In theory, the fingers on the VPL are capable of accuracy up to 1 degree. In practice, 20 degrees or more of crosstalk was not uncommon.

The DataGlove requires calibration because each user will generate different bend angles for a specific posture (hand position). MITRE developed a calibration procedure that requires the users to repeatedly flex the hand. Based on maximum and minimum joint flexion, the raw data is converted to an angle.

2.5 OTHER DEVICES

2.5.1 Voice Recognition

The Voice Navigator is manufactured by Articulate Systems Inc., Woburn, MA. It is a Macintosh application program (with associated hardware) that allows the user to navigate through the pull down menu items by voice rather than with a mouse. As implemented in the MITRE test bed, the Voice Navigator, resident in the Macintosh personal computer, is connected to the Silicon Graphics workstation via a serial RS-232 link.

The Voice Navigator allows a user to train the unit to recognize a user-defined set of words or phrases. Voice devices such as these may be characterized as Buttons. Each separate group of words recognized has the effect of a Button input. Upon recognition of a combination of words, the application initiates some action.

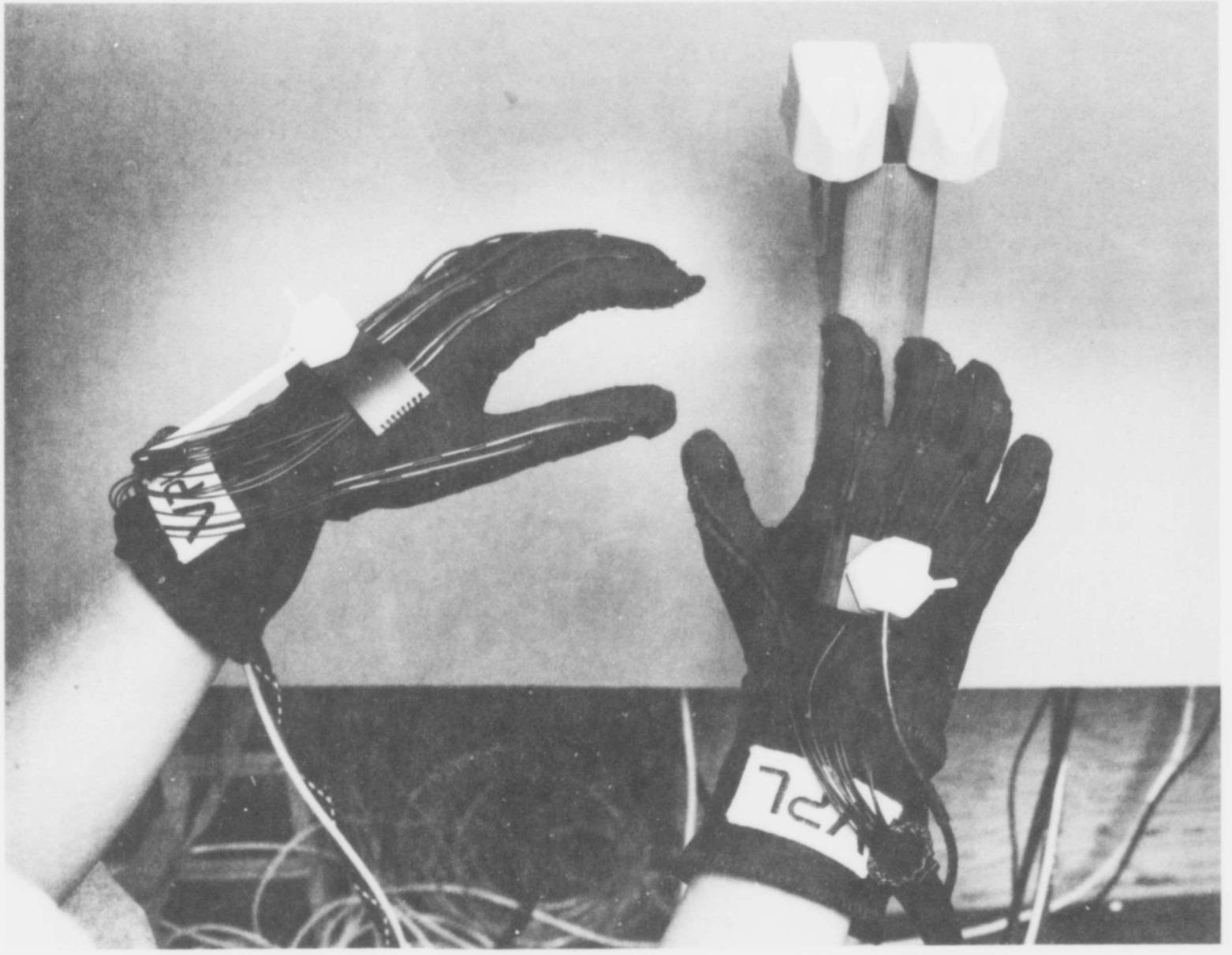

Figure 2.4 VPL DataGlove

2.5.2 Feedback Devices

Positive feedback that something occurred is a very important aspect of VR. Most of today's products provide visual feedback. However, the real world supplies us with audio feedback, tactile feedback, and force feedback.

Auditory Feedback. Auditory feedback might include verbal instructions, or various sound effects. Although we did not formally evaluate a range of audio products, we believe that this is a potentially important area.

Crystal River Engineering makes the Convolotron and the Acoustotron. These devices are special-purpose parallel computers, which model the acoustical properties of the ear for several localized sound sources. These machines provide a very powerful perception of directional sound sources. Headphones are required to fully appreciate these effects.

Another approach to auditory feedback is speech synthesis. This approach has been in user for several years. Some simulators, for example, use speech synthesis to simulate air traffic controller instructions.

Finally, non-speech audio can be used to provide symbolic or representational sound effects. An especially viable approach is to incorporate a Musical Instrument Digital Interface (MIDI). MIDI is widely used to control electronic keyboards and synthesizers. Many modern instruments can control digital samples of *any* arbitrary sound, voice, or noise.

We incorporated this last approach into our prototype application. An Ensoniq EPS-16 digital sampling workstation, connected using a KeyTronics MIDIator RS-232 adapter unit, allowed us to control audio output by sending MIDI command codes to a serial port.

Tactile Feedback. The two tactile feedback devices considered were both mentioned previously. One was the TouchMaster finger buzzers associated with the DHM. The other was the vibration associated with the Global 3D Controller. As of this writing, our evaluation of these devices has been inconclusive.

Force Feedback. Exos Corporation, MIT and others are now experimenting with a variety of products that provided positive force feedback to the user. To date, these products are very much experimental and very expensive. We believe that this is a potentially important area of research.

2.6 HEAD-MOUNTED DISPLAYS

One of the goals in the development of VR is to achieve "total immersion" in which one sees and interacts with objects in a virtual world in the same way that one sees and interacts with the objects of the real world. Developers of VR have utilized the head-mounted display (HMD) as a means of displaying the virtual environment to the user. The HMD has previously been used as an information display in military aircraft. In an HMD, displays and imaging optics mounted on a headset provide a virtual image in front of the eyes. In the VR domain, this design provides the user with a view of the virtual environment while blocking out the user's real environment. Several HMDs are commercially available for VR applications.

This section assesses the present state-of-the-art in HMD technology applied to immersive VR. It describes the technical capabilities of existing HMD configurations. It evaluates these characteristics with respect to goals set below the capabilities of the human visual system and which do not overwhelm current display technology. Thus, it provides salient performance characteristics and limitations that must be considered by a VR systems designer/implementor in order to provide a functional display of immersive VR in conjunction with other VR input/output devices.

2.6.1 Display Requirements

HMDs consist of four basic parts:

- a display or image source, usually a cathode ray tube (CRT) or a liquid crystal display (LCD), which presents an image to the viewer; (For binocular viewing, two displays are required, one for each eye.)
- an optical system that allows the screen to be placed very close to the eyes and head for compactness, magnifies the image so it appears "life size," and ideally provides a wide field of view (FOV);
- a head mount, which provides a base for mounting these components; and
- a head tracker, which monitors the position of the user.

The purpose of using an HMD in VR is to simulate the user's visual experience of the real world. To achieve such an effect, we consider the following characteristics of normal visual sight:

- Binocular/stereoscopic vision
- Field of view
- Resolution
- Color
- Distortion and aberrations
- Accommodation (focus) over a range of viewing distances
- Other important considerations:
  - users differ anatomically
  - worn optical device comparable to eyeglass weight with no tethers

These characteristics are both necessary and desirable for the HMD to provide an effect of total immersion that matches the viewer's natural experience. Realizing, however, that the reproduction of normal vision in an HMD or with any other current display device might be beyond the ability of current display technology, we set minimum goals that will make HMDs usable for VR.

2.6.1.1 Binocular/Stereoscopic Vision

Binocular vision is an important depth cue. A slightly different perspective of an object presented to the right and left eyes is fused by the brain, which enables us to perceive an object in three dimensions. Depth perception provided by binocular vision must be carried over into the virtual environment in order for the user to function naturally in VR. It aides the user in making correct judgments about the relative positions of objects.

2.6.1.2 Field of View

The monocular FOV of the human eye is 150 degrees in the horizontal and 135 degrees in the vertical [Clapp 87]. The total horizontal FOV is 180 degrees (without head motion). Thus there is a natural binocular overlap of about 120 degrees. The FOV of the HMD must approximate this as much as possible for the use of the peripheral field and the illusion of total immersion.

2.6.1.3 Resolution

The human eye is capable of discerning an element size corresponding to about 1 minute of arc at the center of the person's gaze. Although this resolution would be desirable, current display technology cannot provide this resolution, especially in the size of displays typically used in the HMD. A 1280 x 1024 white pixel matrix in a small display is the minimum acceptable addressable resolution needed to take advantage of present commercial work-station image generating capabilities. This corresponds to about 2 to 3 minutes of arc per picture element assuming a 50 degree monocular FOV at center of gaze.

2.6.1.4 Color

Since people are accustomed to viewing the world in color, a full-color display would add to the realism of a system. From current experience with graphics workstations and display performance, a minimum of 15 RGB bits should be required. Therefore, a color palette with 32,768 colors is anticipated from the display.

2.6.1.5 Freedom from Distortion and Aberrations

The environment is perceived by the eyes in an orthoscopically correct manner. The brain compensates for most distortions inherent in the human visual system, providing distortion-free perception. The proportions of viewed objects are realistic and not compressed, elongated, bent, or distorted in some other way. Misjudgments that are possible in a distorted view of an environment might also affect the user's performance in that environment.

2.6.1.6 Accommodation/Convergence

The human eyes accommodate, or focus, and converge to the same point in space when viewing an object in the three dimensions of our natural surroundings. The simultaneous accommodation and convergence of the eyes allows one to comfortably view objects that range in distance from infinity to the individual's near point of accommodation, a distance of about 0.25 meters. Ideally, we would like to have the same viewing range with an image provided by the HMD.

2.6.1.7 Adjustment Elements for User Variation

The physiological makeup of each person is not the same. Therefore, adjustable elements must be incorporated in the design of the HMD to account for differences in head size, interpupillary distance (IPD) to accommodate binocular viewing, and the wearing of eyeglasses (eye relief).

2.6.1.8 Physical Limitations

An ideal HMD for use in VR should weigh no more than an average pair of eyeglasses and be as easy to place on and off. This is not within the realm of current technology. The weight of the HMD must be minimized to avoid neck strain, user fatigue, and gain user acceptance. An acceptable maximum weight for the HMD is about 12 ounces. Counterweights and tethers should be eliminated.

3.6.2 Products Assessed

An HMD is a virtual image display in which the image typically appears to be located at some distance beyond the eyes of the viewer. There are three basic optical designs for current HMDs:

- Infinity Optics — An optical configuration that collimates the rays from an image source to produce an image that appears at infinity.
- LEEP Optics — A pair of wide-angle lenses (in a magnifying configuration) originally designed to view side-by-side stereographic photographs of a scene.
- Fresnel Optics — An optical assembly that has on one surface, elements with variable optical bending (refractive) power in an ordered adjacent spatial arrangement. The assembly replaces an equivalent heavier single focus (thick) lens.

Table 3 compares the natural human visual characteristics to the suggested performance goals for HMD visual and selected physical characteristics. We evaluated five commercial HMDs used in VR on the basis of the technological limitations with respect to providing the goals of performance outlined above. The HMDs are:

1. **Quarterwave** (n-Vision). This HMD uses an infinity optics system with monochrome cathode ray tube (CRT) displays. The largest advantage of this model is that it provides high resolution.
2. **BOOM2C** (Fake Space Labs). This is a CRT-based display with limited color gamut provided by two primaries, mounted on a counterbalanced arm device. The BOOM2C also employs the use of LEEP wide angle optics, manufactured by LEEP Systems, of Waltham, MA. The chief advantage of LEEP Optics is a wide FOV.
3. **Cyberface 2** (LEEP Systems). The cyberface uses full color liquid crystal displays (LCDs) in conjunction with its LEEP optics. It achieves a wide FOV but suffers in other areas, such as resolution.
4. **Flight Helmet** (Virtual Research). The Flight Helmet is similar in basic design to the Cyberface, using LCDs and LEEP optics, but it does not achieve the same FOV for reasons explained below.
5. **EyePhone HRX** (VPL) The EyePhone HRX uses full color LCDs with Fresnel lenses, and not the LEEP optics, in an attempt to improve on the resolution and reduce distortions caused by the LEEP lenses.

Table 3. Suggested VR HMD Goals versus Human Visual Performance

	Binocular Overlap	*Total FOV*	*Resolution*	*Color*	*Weight*	*Tethers*
Human Vision	120 deg	180 deg H 135 deg V	1 arc-min/ element	greater than 1,000,000	N/A	N/A
Realistic Goals	70 deg	140 deg H 100 deg V	2 arc-min/ pixel	32,768 distinguishable	12 ounces	none

2.6.3 Product Assessment

We have found that there are inherent tradeoffs with what can be achieved with the technologies implemented in the HMD. All the parameters cannot be simultaneously achieved in one HMD, as we will see below.

2.6.3.1 Obtaining a Large Field of View

The field of view (FOV) of the HMDs is a function of several factors. The optics themselves can play an enormous role in expanding a rather narrow field of the display into a large FOV, as with the LEEP optics. The FOV is also affected by the size of the display used, and the size of the exit pupil of the optics.

Because of its size, usually on the order of 0.5 to 1 inch diagonal, the CRT display is not conducive to a wide FOV. A larger CRT, which could provide a significantly larger FOV, is not feasible because of the weight added to the HMD. The size of the exit pupil of the optics, about 12 mm, combined with an eye relief of about 30 mm needed to accommodate eyeglasses, further decreases the FOV.

Magnifier optics can be constructed to easily produce a wide FOV. In the case of LEEP optics, the FOV is increased by using a large 69 mm diameter lens system. The LEEP Systems Cyberface uses a 4-inch LCD to make full use of the visual field of the optics. The Cyberface obtains a 140 degree total field in the horizontal direction. Other applications of the LEEP optics use a 3-inch LCD, which does not fill out the potential FOV of the optics [LEEP]. The position of the optical axes affects the amount of binocular overlap and thus the region can be viewed stereoscopically. Setting the optical axes parallel (or convergent) reduces the FOV and increases the binocular overlap region. For example, the n-Vision HMD provides a narrow 50° FOV with a 100% or 50° binocular overlap. The Cyberface 2 claims a total FOV of 140 degrees. It has a 25 degree divergence in the optical axes, which maximizes the FOV at the expense of the region of binocular overlap.

LEEP optics are used with 4-inch diagonal CRT displays in Fake Space Labs' BOOM counter-balanced arm device. This allows the wide FOV to be used in conjunction with the high resolution image. The use of LEEP wide angle optics with CRT displays in an actual HMD is contraindicated by the fact that a small 1-inch display would not fill up the visual field of the optics.

2.6.3.2 Resolution

The resolution capability of an HMD depends on the type of display used, CRT or LCD, color characteristics, and the addressable and visual resolutions of the display. The magnification of the displays by the optics also plays a significant role in the achievable resolution. Once again, other performance issues play a role in determining what is feasible.

Current CRTs used in HMDs provide 1280 x 1024 addressable elements in a 1-inch diagonal monochrome display. The CRT display used in the n-Vision HMD yields an addressable resolution of 2.3 arc-minutes horizontally by 2.9 arc-minutes vertically per pixel.

A LCD display, as used in LEEP Systems Cyberface 2, is a 4-inch diagonal full-color LCD that renders a white pixel matrix of only 319 x 117 (using the methodology of Dunn-Roberts 92). A resolution of 21 arc minutes horizontally by 44 arc minutes vertically per white pixel is obtained. These are exceptionally large. The pixels at the edges of the screen are even larger due to the nature of the distortion of the optics.

The VPL EyePhone HRX provides a resolution of about 12 arc minutes horizontally by 16 arc minutes vertically per white pixel element. The resolution of LCDs, when compared to the high resolution capability of the eye, represents the greatest problem with LCD-based HMDs. LCD-based HMDs make wearers "legally blind," 20/200 on the Snellen Eye Chart [Taber 89]. With a CRT, the resolution is enhanced by a factor of at least five.

2.6.3.3 Color

As indicated above, the use of a color display significantly reduces the ability to achieve a good resolution, since the number of individual color elements needed is tripled to achieve the same addressable resolution in terms of white picture elements.

2.6.3.4 Tethers and Weight

The weight of the HMD is an important factor affecting the performance issues regarding the use of the LCD versus the CRT.

The physical makeup of the CRT, however, will not allow further reduction in size and weight to make it applicable for use in VR HMDs. A miniature CRT assembly slightly less than 1 inch in diameter weighs between 3.5 and 6 ounces and is at least 4 inches in length [Hughes]. The LCD, on the other hand, has a relatively small weight (2.8 to 6.3 ounces) for the larger 3-inch to 4-inch screen and is less than 1 inch thick [Sharp]. The LCD resolution, however, is no better than one-third of the resolution of the CRT. Unfortunately, there is not a large enough commercial demand for higher resolution LCDs to warrant their further development for this particular application.

Tethers are unescapable with an HMD because present designs require cables for power and signals for the display media. It is desirable to minimize the confining effects associated with the use of tethers.

2.6.3.5 Other Technical Issues

In general, we have found that there are certain technical issues that impose strict limits on the visual performance of HMDs and impose challenging constraints to overcome in future HMD designs for VR use.

Associated with the binocular viewing of stereoscopic images is the accommodation/ convergence conflict. In the real world, the eyes accommodate and converge to the same point. The lack of conflict in natural viewing allows us to view images clearly over a large range of distances. In stereoscopic viewing, on the other hand, the eyes no longer accommodate and converge to the same point. The eyes can comfortably tolerate only a certain amount of discrepancy between accommodation and convergence angles [Lipton 82]. This effectively sets limits on the viewing range of the display unit, which depend on the placement of the virtual image plane. Application of some VR devices, such as the DataGlove, at close range in virtual space is difficult. The rendered computer generated VR environment must be designed with these constraints in mind. Geometric models that provide for the comfortable and accurate perception of computer generated stereoscopic images have been derived [Southard 92]. In addition, the computer generated images presented should have correct depth cues of perspective size and interposition.

It is impossible to overcome the accommodation/convergence conflict in a binocular optical system where the virtual image plane is at a fixed distance from the user. Only through a variable virtual image plane system will the conflict of accommodation and convergence be resolved so that viewing can take place over a normal range of distances.

The interpupillary distance (IPD) is a user specific characteristic, which should be taken into account in HMD design. In some systems where the exit pupil of the optics is small, it is necessary to adjust the IPD to fit each user. In the case of the LEEP optics, the exit pupil is large enough to accommodate virtually all differences in user IPDs. In any case, since the

perception of depth is specific to each person's IPD, it is necessary to introduce into the graphics rendering model the IPD as a user specific constant [Robinett 91]. This eliminates the possibility of divergence of the eyes in fusing the images. LEEP Systems has also already taken this into account, as the spacing between the images at infinity (62 mm), is slightly less than the interaxial spacing of the lenses (64 mm).

2.6.4 Results

The performance of commercial HMDs is summarized in Table 4. The HMDs fall short in being able to simultaneously achieve all of desired operating characteristics. However, understanding the visual performance limitations of the HMD, the VR systems designer should be able to finesse the VR presentation with appropriate graphics rendering models, and the appropriate inclusion and use of available VR input/output devices.

Table 4. Comparison of Specification of Existing HMDs

	n-Vision	*LEEP*	*Virtual Research*	*Fake Space Labs*	*VPL*
Model	Quarterwave	Cyberface 2	Flight Helmet	BOOM 2C	EyePhone HRX
Optical System	Infinity	LEEP	LEEP	LEEP	Fresnel
Display Device	CRT	LCD	LCD	CRT	LCD
Total FOV (Degrees)	50-83 H/50 V	140 H/95V	100 H/66 V	140 H/90 V	106 H/75 V
Addressable Resolution*	1280x1024	479 x 234	360 x 240	1280 x 1024 1280 x 512	1280 x 512
Resolution (arc-minute)***	2.3 H x 2.9 V	21 H x 44 V	19Hx28V***	4.2 H x 5.3 V	12 H x 15 V
Color	Mono	Color	Color	Mono 2 Color	Color
Weight	6.0 lbs	2.5 lbs +	3.7 lbs	N/A	2.5 lbs
Cost	$60K	$8.1K	$6K	$74K	$49K
Binocular Overlap Degrees	50-68	50	32	50	60

* Reported addressable resolution

** For LCDs the resolution follows Dunn-Roberts' calculations and refers to the white picture elements

*** Based on the assumptions that the monocular FOV is 66 degrees as given in the LEEP standard product guide

\+ The Cyberface 2 weighs 4.25 lbs with the Counterpoise, a counter balancing device.

2.7 SUMMARY

The input devices available today would have to be rated as both crude and limited for VR applications. Range, tethers, and co-site interference limitations are among the biggest problems. Because of these limitations, the user does not immediately interact with the virtual environment in a complete and/or intuitive fashion. There is a learning curve that the user must experience. It depends on the application, the device(s) used, and how the devices complement each other in a specific application.

For example, in one demonstration, a user wearing one VPL DataGlove was able to reach for, grab and move a virtual cube that he/she visualized in real time on a large screen stereoscopic display. The demonstration was very effective because the user interface is sufficiently straightforward that its use required minimal learning time. This allowed a person to whom we are demonstrating this technology to get a hands-on experience. This is important, because the effectiveness of VR is the power of the immersion and the sense of presence and control one feels. That sense cannot be attained by merely watching someone else.

A more complicated demonstration allowed a single-user wearing two VPL DataGloves to reach for, grab, move and throw a virtual cube with either the right or left DataGlove. The cube was then passed back and forth from hand-to-hand, thrown or caught. This was somewhat more difficult, but the user quickly became adept with a little practice.

Because of the combination of the real-time 3D graphics presentation and visual feedback provided to the user with the VR devices, the degree of immersion and sense of presence and control attained by the user was markedly enhanced. Therefore, in spite of the physical limitations involved with the present state-of-the-art VR devices, a user can readily become accustomed to VR with the choice of a well designed application. The quality of the application will depend, in a complicated fashion, on the graphics, the devices, the user interactions and the software that ties it all together.

Generic portable software that is device independent should allow for significant improvements to the hardware with minimal impact to the software. For example, if a good algorithm for representing a hand is developed, there is negligible impact on the rendering software if the hardware becomes better quality. The only modification required is to transfer position and finger data from the new "hand-like" devices to the rendering software.

For the work described in this report, we selected the n-Vision monochrome CRT-based HMD because it provides the highest visual resolution and is compatible with the high resolution modes of operation available from our SGI workstation. For development of software/hardware for the VR platform, we also utilized a stereoscopic large screen display

(LSD). The LSD consists of two front projected Electrohome ECP-4000 color projectors (see figure 2.5). The optical output from each projector is polarized orthogonally with respect to the other. The user wears a corresponding set of polarized glasses to separate the left-eye and right-eye views for stereoscopic viewing. The multi-channel option (MCO) board from SGI provides the high resolution stereoscopic capability for simultaneous use with the n-Vision HMD and the 3D stereoscopic LSD. The LSD is used mainly for group viewing of VR demonstrations and prototyping. For this configuration, user feedback has been extremely positive.

Figure 2.5 VR Laboratory

SECTION 3

SOFTWARE

In this section we consider the principal functional requirements of a VR environment and the implications which these requirements have for the software architecture. We draw from examples of several VR architectures that have been implemented by researchers and developers to illustrate how these requirements can be realized. As a final example, we describe the MITRE Virtual Environment Architecture (VEA), which we have developed.

3.1 FUNCTIONAL REQUIREMENTS FOR A VIRTUAL REALITY

A multisensory VR imposes unique requirements on the interface software. This software must support the simultaneous use of multiple input and output devices and provide real-time response that is sufficient to avoid disturbing latency or discrepancies among the output modalities. To be usable, the software must facilitate the integration of applications with the VR interface. It must be flexible and extensible to minimize the impact of adding additional functionality or devices.

3.1.1 Response

The simplest conceivable VR environment would comprise a single-user interaction device, a processing element containing the application and model of the virtual world, and a graphics rendering and display capability to provide a view into this model. The interaction device could represent, for example, a DataGlove or a position tracker affixed to a head-mounted display. When the user takes an action, the software must acquire the new input state, make the appropriate changes in the state of the model or application, and redraw the graphical representation of the model to be presented to the user. As the user must react to the view he is presented, the entire process forms a feedback loop as shown in figure 3.1. System usability depends critically on the latency or delay which occurs between a given user action and the graphic response of the system to this action. There is no widespread agreement on acceptable latency for VR systems. Flight simulator designers have suggested 60 milliseconds as an upper limit, but any such limit is dependent upon the rate of user interaction with the system. Blanchard [92] suggests that systems exhibiting a latency as great as 200 milliseconds could be usable and convincing in some applications. The same authors assert that consistency of response time is even more important than response time itself.

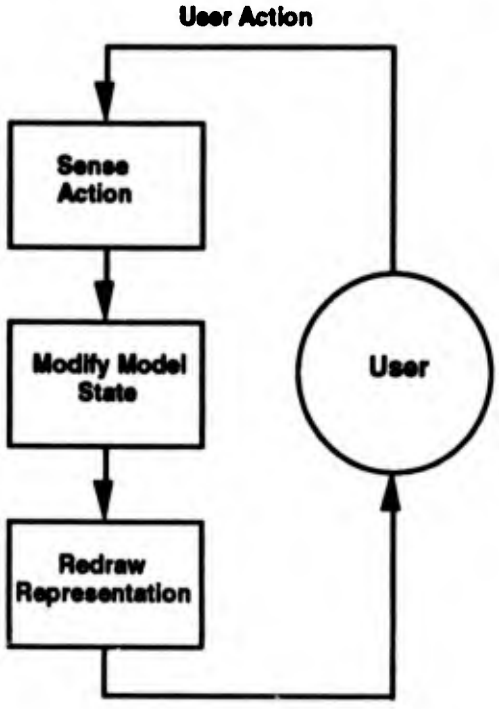

Figure 3.1. Single User-interaction Device FeedBack Loop

3.1.2 Multiple Devices

Much of the promise of VR lies in its ability to integrate multiple inputs from a user and to provide multisensory feedback. Practical VR systems, therefore, must be capable of processing simultaneous input and output streams. The system software must not only respond with minimum latency, but must also maintain consistency among the responses. Human observers are especially sensitive to disparities between stimuli, and poor synchronization can induce nausea [Ellis 91].

3.1.3 Distribution

In the near term, the distribution of processing tasks over multiple hardware platforms represents one means of obtaining the needed system response. One of the most promising future application areas for VR is collaborative planning or design. To accommodate these applications, the VR system software must be capable of running concurrently on multiple processing environments. Such environments can potentially be heterogeneous and geographically separated, connected by a wideband network. Our effort was concentrated on the development of a single-user application. However, the architecture is extensible.

3.1.4 Ease of Integration

VR is becoming increasingly viewed as a flexible interface technology which can be applied to a wide range of new and existing applications. Because VR software environments are complex and expensive to develop, they should be designed for maximum portability among applications. Ideally, the software architecture of a VR should accommodate the integration of separate autonomous application code, written in heterogeneous programming languages.

3.1.5 Ease of Extension

At the current stage in the evolution of VR, new interface devices and capabilities are being introduced at an ever-increasing rate. For the foreseeable future a substantial share of VR implementations will be dedicated to the evaluation of the capabilities of these evolving devices. As a consequence, the software architecture of VR systems must be designed to accept a variety of input devices without substantial additional software development.

3.2 IMPLICATIONS OF FUNCTIONAL REQUIREMENTS

Each of the VR software architectures discussed here addresses the above functional requirements to some degree. A comparison of these and other VR implementations reveals several common design approaches, which are enumerated below.

3.2.1 Object Orientation

Extensibility and ease of integration with application code demand that VR software environments be highly modular. Thus VR systems are an excellent application of the object oriented paradigm in which there is a one to one correspondence between modeled objects and software entities. In this paradigm, adopted in various forms by most VR developers, each object encapsulates its own state and the means by which other objects interact with that state. This is especially important when objects in the virtual environment are distributed over multiple hardware platforms. Moreover, object oriented architecture provides a mechanism for encapsulating an object's *behavior* — that is, how it reacts to changes in the environment and other objects. Encoding of behavior within objects is essential to the provision of autonomy and verity in a VR.

3.2.2 Parallelism

The single-threaded sense-compute-display loop depicted in figure 3.1 becomes inadequate when additional input and output devices are added. Sequential processing of data from multiple input devices and to multiple output devices creates potential problems

with synchronization and consistency. As a consequence, most VR developers have adopted mechanisms for parallelism. As a minimum, most designs provide separate concurrently executing server processes for each interaction device. This concept can be extended by providing separate processes for each modeled object or group of objects. This parallelism can be implemented at varying degrees of granularity: with "lightweight" processes executing in a shared data space, as full processes executing on the same platform, as processes distributed over multiple, potentially heterogeneous and geographically separated platforms, or a combination of these.

3.2.3 Asynchronous, Message-Based Communication

Encapsulation of objects and coordination of object processes executing in parallel require carefully designed inter-object communication. Message passing is the most commonly used mechanism for this communication because it readily supports distribution. Most VR designers have favored asynchronous message passing, in which the sending process continues to execute and does not block to wait for a response. Message processing architectures may provide for direct process-to-process communication, broadcast communication (in which the sender does not know the recipients) or a combination of these.

3.2.4 Layering of Device Abstraction

All VR architectures require software modules to acquire the data streams from interaction devices and convert this data into a form which is meaningful to the interaction modality. This code is, by necessity, specific to each interaction device. To minimize the development impact of the integration of new devices, many VR developers have divided device server software into device dependent and independent layers. This permits the abstraction of interaction devices into classes such as locators, gestures or speech. The higher layers may thus be shared among multiple devices and only the device-dependent layers need be rewritten to integrate new modules.

3.3 REPRESENTATIVE ARCHITECTURES

This section compares four implementations of VR architectures: IBM's Veridical User Environment (VUE); the Virtual Environment Operating Shell (VEOS) developed by the Human Interface Technology (HIT) Laboratory at the University of Washington; dVS, a commercially available VR system from Division, Inc.; and the Virtual Environment Architecture (VEA) under development to support VR prototyping and research at The MITRE Corporation.

3.3.1 IBM Veridical User Environment (VUE)

The IBM VUE system [Appino 92] decomposes the virtual environment into multiple client/server processes which may be executed in parallel on heterogeneous platforms. Figure 3.2 depicts the VUE architecture. Device and application servers communicate with a single client, the Dialogue Manager (DM) using asynchronous message passing. A single device server process exists for each interaction device which comprise the device-dependent layers of interaction processing. The application process or processes encapsulate the objects modeled in the VR. To improve responsiveness, application processes can communicate directly with one another where there is need to transfer large volumes of data, as in the transfer of object representation data to the renderer.

The Dialogue Manager (DM) serves the function of coordination among the multiple interaction device servers and the application processes. This connecting layer is an event driven, rule based architecture which comprises a message interface to server processes and a dialogue. The dialogue consists simply of a set of rules of the form [Appino 92]:

INCOMING_EVENT —> RESULT_EVENT

Events consists of an identifier and carry an associated message packet. When the DM receives a message from a server, it is packaged into an appropriate event. A rule matching kernel in the DM compares the event with its rule base and "fires" the appropriately matching rules, if any. The firing of a rule can result in the modification of the message packet associated with the INCOMING_EVENT for encapsulation in the RESULT_EVENT. Optionally, the firing of a rule can have a side effect which modifies state variables within the DM.

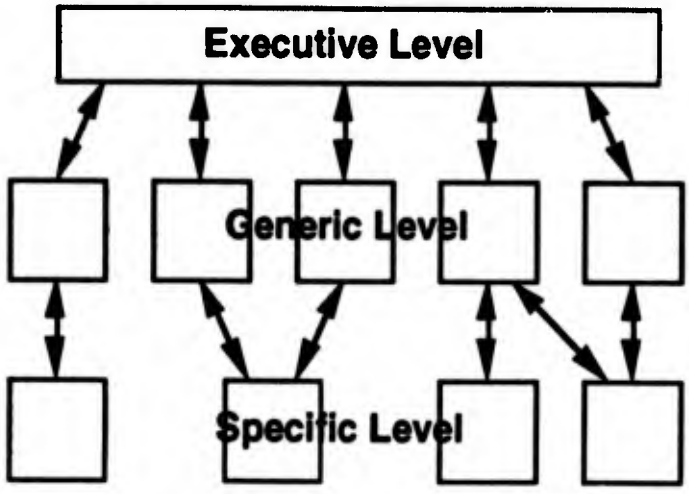

Figure 3.2. IBM's VUE Software Architecture

The rule structure of the DM is partitioned into hierarchical layers as shown in figure 3.3. Specific level rules encapsulate the process of dealing with data from a particular device. Generic level rules accept events from the specific level and interpret them in terms of more general interaction paradigms such as moving an object or viewpoint. Executive level rules are used to define the available modalities of interaction with the application processes and associated objects in the virtual environment. Within each layer in the hierarchy, rules can be aggregated into rule sets known as subdialogues which encapsulate an associated data state. Rules within a set can modify the state of data encapsulated within that set.

Events are propagated between layers using an asynchronous broadcast form of message passing such as described by Hill [92]. In this model, the sending rule does not know the recipient of an event; there may be zero or more rules which match a given event [Appino 92].

The asynchronous message passing model increases the logical independence among servers and among subdialogues within the DM, making VUE easy to distribute among heterogeneous platforms. In a fluid flow visualization developed by IBM J. Watson Research Center, the VUE was distributed over a Stardent Titan and two IBM RISC System 6000 workstations on an ethernet network. The Titan workstation hosted the fluid dynamics simulation servers and, because of its high speed graphics capability, the rendering server. One System 6000 workstation hosted the Dialogue Manager, and the second System 6000 hosted the glove, tracker, sound and speech servers.

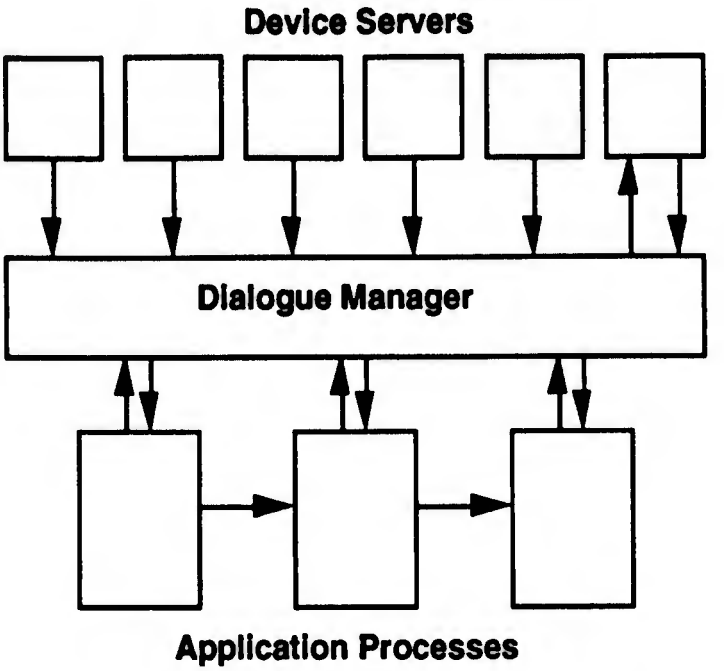

Figure 3.3. Partitioning of Dialogue Manager

3.3.2 VEOS

The Human Interface Technology Laboratory at the University of Washington has developed an extensible environment for developing distributed VR applications. This environment is designated the Virtual Environment Operating Shell, or VEOS. The following discussion is based upon the documentation of VEOS version 2.0 [Coco 92].

In a VEOS system, the virtual environment is partitioned into object-oriented clusters of data and associated computational tasks known as "entities." Each entity executes under a copy of the VEOS KERNEL in a separate UNIX process. As in VUE, inter-entity communication is by means of non-blocking, asynchronous message passing, which is implemented using Berkeley UNIX sockets. Distribution is straightforward; entities can execute on the same or networked UNIX platforms. The design of VEOS is flexible and symmetric, and does not force any particular model for distributing the functionality within the virtual environment, such as the layered client/server model of VUE. Efficient designs require, however, that the entities be as self-reliant as possible to minimize the impact of interprocess dependence.

Within each entity, the VEOS kernel provides three principal services: generalized data management, inter-entity communication, and interpretation of the entities' coded task description. Each entity contains a database manager known as *Nancy* which manages data in the form of "grouples." A grouple is defined as a nestable tuple, conceptually equivalent to a Lisp list. Access to data in *Nancy* is through a pattern matching paradigm, rendering it effectively a content addressable database. Locations within the database, known as the "grouplespace" are found by specifying a pattern to match. VEOS includes a pattern matching language which provides for wild cards and iteration over elements within a grouple. Once a data location has been addressed by pattern matching, the fundamental operations provided are insert data, retrieve data and replace data.

Asynchronous message passing to other VEOS entities is supported by a component known as Talk. Messages consist of data in linearized grouple format. Talk provides two simple message passing primitives, *vthrow* and *vcatch*. The primitive *vthrow* sends a message to one or more entity destinations. Note that this is not a broadcast model of message passing; destinations must be known and expressed in terms of the entity host platform and port number. Incoming messages are enqueued, and *vcatch*. returns the oldest available message. Incoming message grouples are evaluated as Lisp expressions. These can contain function calls to member functions contained within the entity description and the associated arguments to these functions.

Each VEOS entity executes a single-threaded processing loop or "frame." At the beginning of the frame control is passed to the kernel to do housekeeping functions such as memory management and network maintenance. Next *vcatch* is called iteratively to retrieve enqueued input messages. Each input message is unpacked and evaluated using the Lisp *eval*. Finally, a list of zero or more background functions associated with the entity is called iteratively. Although each entity executes within a single UNIX process, these functions are used to simulate concurrent background processes if they are designed to execute a short, non-blocking computation and quickly return.

The above-described features make VEOS a highly generic software framework for the development of distributed, object oriented systems. It must be noted here that VEOS is a research vehicle which is under continuous development and therefore is continually evolving.

3.3.3 Division dVS

In contrast to VEOS, dVS is a commercial product marketed by Division, Inc. dVS is designed as a highly parallel VR development environment to execute on the Division Provision™ machine. Because Provision™ is based on a distributed memory parallel machine architecture, dVS is inherently a distributed environment. Provision™ partitions the

hardware environment into autonomous processing clusters, each of which has its own local memory and dedicated processing elements. In dVS, the software environment is partitioned into parallel processes known as *Actors*, each of which provides a related set of services or models an aspect of the VR. Each *Actor* executes on its own Provision™ processing cluster, and all *Actors* are coordinated by a special actor called the *Director*.

The fundamental components of a dVS VR are called *Elements*. These are abstract data types which can be instantiated to encapsulate the state of objects in the VR. In dVS version 0.1 four basic *Elements* are supported: lights, environment_objects, cameras, and controls. A light defines a single point light source for rendering purposes. Environment_objects are data types which define the attributes of a 3D object in the VR. The camera *element* encapsulates the position and other attributes of the view volume. Controls are used to hold data from interactive input devices.

An *Actor* creates an instance of an *Element* in order to maintain the state of an environmental object. Access to an instance of an *Element* is not limited to the *Actor* that created it. Each *Actor* that registers interest in a given *Element* is granted access to all instances of that element. This is implemented by each *Actor's* holding a local copy of the instance which it is free to modify. When an *Actor* wishes to commit the state change to the entire VR environment, it sends a message to the *Director* which coordinates the updating of all instances of the *Element* held by other *Actors* in the environment. The *Director* maintains a list of the *Elements* in which each *Actor* is interested, and provides access control to the instances to insure data integrity. Once an *Actor* has declared interest in an *Element*, it must explicitly call a function to be informed of updates to instances of that *Element*. The request for update can be either blocking or non-blocking as desired. Thus either synchronous or asynchronous broadcast message passing can be implemented. The process is shown schematically in figure 3.4.

To support VR application development, Division Inc., supplies a library of standard *Actors* with dVS. These include modules that perform the device specific functions associated with obtaining information from interaction devices. Interface *Actors* obtain information from hardware devices such as DataGloves or head trackers and create instances of a control *Element*. Other *Actors* can register interest in control *Elements* and obtain access to the input data for higher level processing, thereby implementing the layering which is necessary for extensibility in VR. Another dVS library *Actor* provides the developer with a toolbox which implements higher level interaction functions such as walk, fly, and rotate_environment.

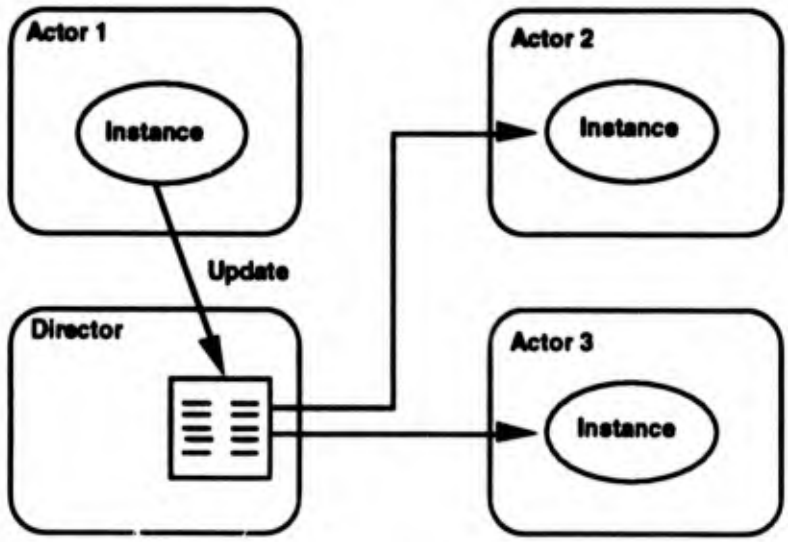

Figure 3.4. dVS Architecture

3.4 MITRE VIRTUAL ENVIRONMENT ARCHITECTURE (VEA)

The Virtual Environment Architecture (VEA) is under development at The MITRE Corporation for use as a development framework for ongoing VR prototyping and research. VEA is a flexible, object oriented architecture which is designed to support the essential VR software attributes enumerated earlier in this paper. The principal components of VEA are shown in figure 3.5. The heart of VEA is the kernel which provides basic services to the environment:

- Loading and saving of environment configuration files
- Synchronization of object processes
- Event-based inter-object communications
- Real-time clock with time expansion/compression

Another major component of VEA is the application itself. For our prototype environment, the application is implemented through the use of a knowledge base. The knowledge base consists of a collection of objects (e.g., tanks, ships, aircraft, etc.,) that make up the virtual world. The knowledge base runs as a separate process independent of the kernel. The kernel and the knowledge base communicate through an event mechanism. The knowledge base has as its interface, a manager that receives events from the kernel and distributes each event to the appropriate knowledge base objects. In addition, the manager is also responsible for sending events generated in the knowledge base to the kernel.

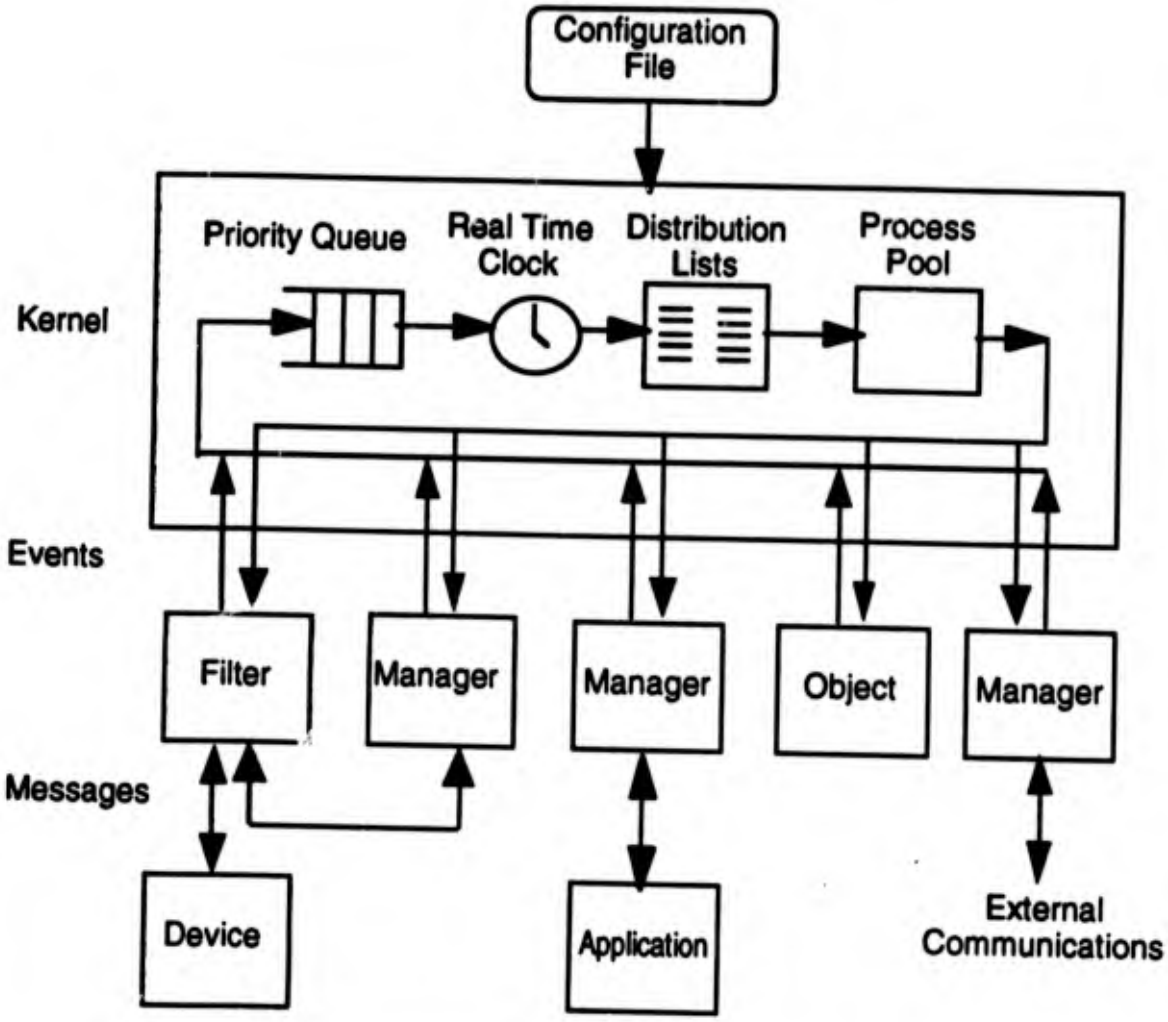

Figure 3.5. VEA Architecture

In contrast to the architectures described, VEA supports two distinct models of inter-object communication. *Events* are asynchronous broadcast communications which are used when the intent is to effect a global state change in the environment. *Events* carry an associated time of occurrence. *Messages* are used for direct, object to object communication.

When an object is initialized, it subscribes to the types of events that it wishes to receive. This is recorded in a distribution list stored in the kernel. As events are generated by objects, they are placed on a queue prioritized by time stamp. The kernel maintains a real time clock and dispatches events at the time of "occurrence." This unique feature allows a user to compress or expand the rate at which the VR executes. All objects on the distribution list for an event of a given type receive a pointer to that event. Each object is free to process the event according to its own behavior representation. When all objects have processed an event, it is deleted.

The availability of two message passing models permits the choice of the optimum mode for a given communication. The event model is ideal for effecting a global state change because multiple objects can process an event simultaneously. Furthermore, a resource access control mechanism, built into VEA objects, allows parallel processing of messages.

To facilitate the integration of multiple input devices, VEA uses a layered approach similar to that of previously described architectures. *Filter* objects handle the device-dependent processing, and may be configured to tailor a device to a particular user or to filter unwanted noise from the input stream. *Manager* objects are servers which abstract these inputs into high-level events and broadcast these events to the objects in the VR. Messages are used to communicate between *Filters* and *Managers*, while events are used to communicate between *Managers* and other VR objects. *Manager* objects may also be autonomous application processes which interact with the VR as servers and communicate using the event mechanism. These servers may also be communications ports, permitting the distribution of the VR over multiple processing platforms.

A more detailed description of VEA is provided in section 4.

SECTION 4

A VIRTUAL ENVIRONMENT ARCHITECTURE

In this section, we discuss the MITRE Virtual Environment Architecture (VEA), which has been developed at MITRE to provide a flexible research environment for the development of VR applications. VEA can support a variety of user interface devices, which assists rapid prototyping of highly interactive environments. The following section provides an overview of the VEA architecture. Succeeding sections discuss in more detail the specific design choices we made.

4.1 OVERVIEW

Section 3 and [Masterman 93] analyze the functional requirements for virtual environments, and the solutions offered by representative systems. In summary, the functional requirements are: interactive response times to user inputs; multiple user interface devices, with multiple modes of input and output; distributed processing; easy integration of application code; and extensibility to new interfaces and applications.

As a consequence of these functional requirements, several themes recur in each architecture: object orientation, parallelism, asynchronous message-based communication, and layering of device abstraction. In these respects, VEA shares much in common with other architectures. Essential to VEA is a *kernel.* The kernel is the central core set of functions that implements event handling, a real-time clock and multiprocessing.

- *Object Orientation.* VEA is implemented in C++. The devices and applications are treated as autonomous objects. Most system protocols are implemented high in the class inheritance hierarchy, so that object instances derive most of their systems behavior from their base classes.

- *Parallelism..* VEA assumes a tightly coupled, shared memory, multiprocessing model of computation. In this respect VEA differs from most other architectures, which assume a looser configuration, distributed over a local area network. VEA's approach is consistent with current super graphics workstation architectures (as exhibited by Silicon Graphics, for example), as well as our perception of architectural trends. Network distribution can be supported in the future by adding network adapter software modules.

- *Message-Based Communication..* VEA defines two modes of communication: *messages* and *events.* Messages are synchronous (blocking); events are

asynchronous (non-blocking). Both messages and events can be processed in parallel.

- *Layering of Device Abstraction.*. VEA defines two layers for device abstraction: *filters* and *managers*. Filters provide an interface between raw I/O devices and generalized interactive modes. Filters can perform device-specific optimizations, such as buffering data, removing unwanted sensor noise, or using predictive filters (e.g., to compensate for system latency) on the data. The filters can encapsulate the data into an event, and post the event with the kernel, or pass the filtered data directly to an associated manager for further disposition. Managers transform modal interactions (e.g., glove, pointer, speech) into high-level commands and actions (e.g., gesture, text). They may apply higher-level optimizations, such as parsing groups of events into a single event. Finally, device filters and managers provide the basic functions needed to interact within a virtual environment.

Most of the discussion about VEA will center on events. Events are the primary mechanism for transmitting global information through the virtual environment simulation. The *kernel* is the central core set of functions that implements event handing, a real-time clock, and multiprocessing.

We discuss each of these topics in the subsequent sections.

4.2 EVENTS

VEA uses *events* for representing certain kinds of simulated events and communications between simulated objects. Events have properties that distinguish them from the more general concept of messages. Object-oriented languages, such as C++, the implementation language for VEA, support messages as member functions, or *methods*. VEA contains additional facilities that support events.

4.2.1 Definition

Events answer the question "what happens?" whereas messages answer the question "how does it happen?" *Events* are characterized by the following attributes:

- The primary purpose of an event is to effect a global state change.
- Events occur in time. Conceptually, all objects receive events simultaneously.
- An event can be processed by any object that registers to receive it. Each object interprets an event as appropriate to its function. A sender does not know a priori who will receive its event.
- Events may be initiated by the users of the system, by simulated objects, or by external influences.
- Events are not elements in a communication dialog or protocol.

In contrast, *messages* are distinguished by the following traits:

- Messages result in state changes local to the sending and receiving objects.
- Messages do not occur in the simulated time frame of reference.
- The sender of a message knows which objects should receive the message.
- Messages arise as a consequence of events.
- Messages may be subject to semantic restrictions, as part of a dialog, protocol, or sequence.

We will not discuss messages further, because they are implicitly provided in the C++ language. The rest of the discussion will focus on events.

4.2.2 Types of Events

Event types include: button, gesture, graphic, locator, pointer, position, posture, sound, speech, text, valuator. These also correspond to abstractions introduced in Section 2, table 1, that describe VR device categories. There may be some overlap between event types. For example, a posture event could be represented by a kind of button event. For convenience, however, a separate posture event allows a manager object to monitor sequences of postures for possible gestures. It is hard to imagine gestures from a button box.

Most of these events are concerned with user interaction. However, the user or any simulated object can initiate an event. More types of events will be defined as application requirements dictate. There is, for example, a *trajectory* event which is an event for a particular application internal to VEA (section 4.4.2.5 and 6). The intention, however, is to keep the number of events small, and to keep their meanings as flexible as possible.

4.2.3 Example: Collision Detection

When an object changes its location, based on its internal motion model, it will post its new location with a position event. All spatial objects will then check their collision detection model to see if a collision is possible. Any objects in proximity will then initiate a sequence of messages, which we can term the collision protocol. The proximal objects will know the object involved from the event. The "collision" could take on any number of forms, depending on the objects that are interacting.

For example, consider two simple spherical objects that collide. They might simply be deflected off each other, or perhaps one would stick to the other. In either case, this would affect each object's trajectory. Another possibility is that an object could be causing a force field. Objects entering this field would then be subject to a force, even though there is no direct contact.

In these examples, the original position event leads to a close interaction, in which only the affected objects are communicating with messages. The final result of this interaction will be the posting of new events: new position events that reflect the new trajectories of the objects involved. The simulated environment does not need to know how these interactions take place, but the results will be known by virtue of the new events that are posted.

4.3 THE KERNEL

The VEA kernel uses UNIX system facilities to implement its event-handling mechanisms. Figure 4.1 provides an overview of the event mechanism in the kernel. Events and objects are initialized from a configuration file. Events are placed in a priority queue, ordered by time. Objects are placed in the object list. When an object is initialized, it subscribes to the types of events that it is capable of accepting. The subscriptions are placed in the distribution lists, which are organized by event type. Objects may subscribe, or drop a subscription, at any time during the simulation.

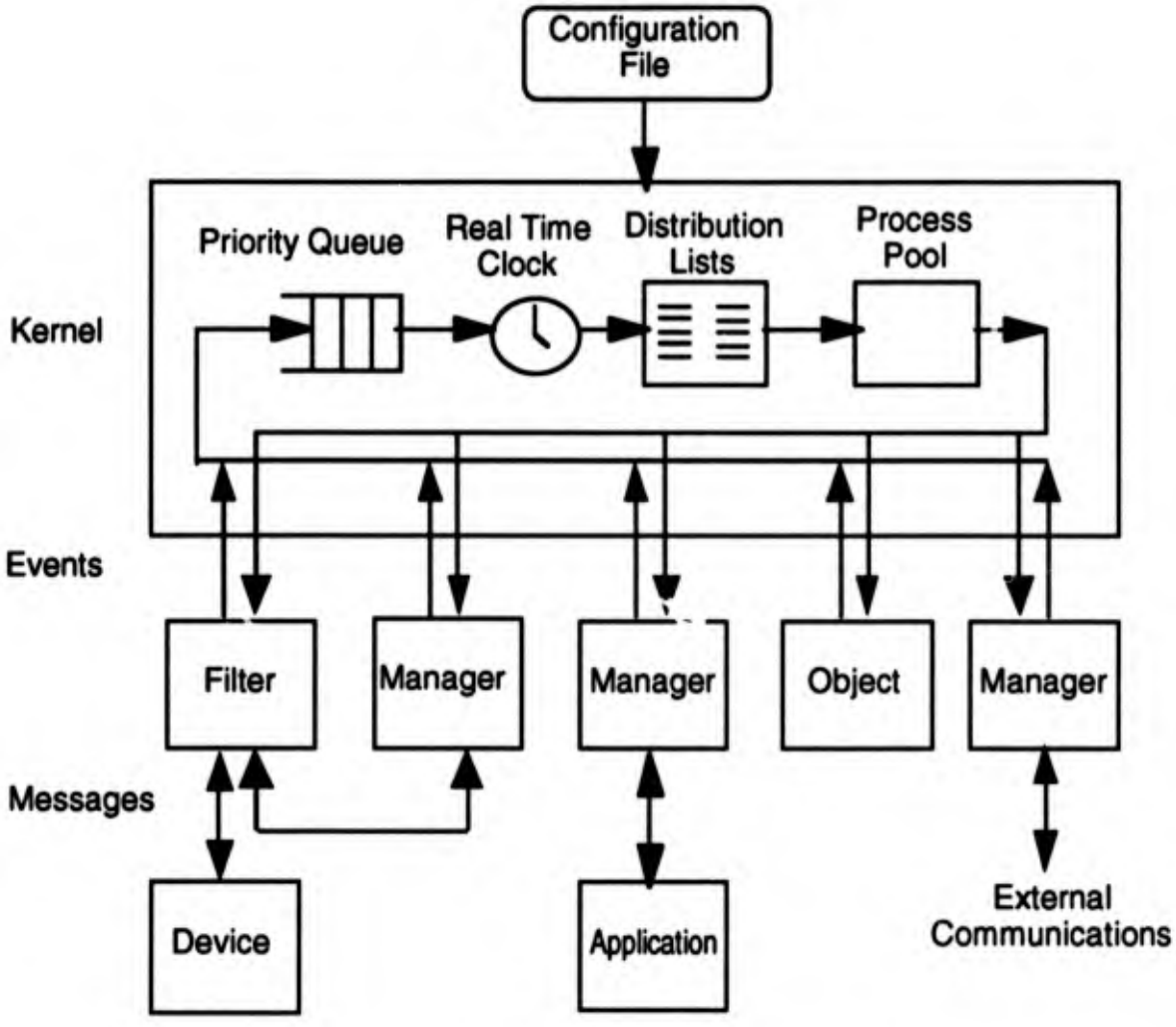

Figure 4.1. VEA Architecture

4.3.1 Event Handling

VEA contains a real-time clock, which is implemented with the UNIX interval timer. A time expansion factor is provided, which allows the simulation to run faster or slower than real-time. Most objects will not need to access the clock itself. The objects usually work with the time stamp reported in the events that they receive.

When the simulation clock reaches the time indicated by the event at the top of the priority queue, the event is dispatched for distribution to the objects. All objects on the distribution list for that event type receive a pointer to the event. Each object can then

process the event, according to its own interpretation. A process from a central process pool is assigned to each recipient for execution of that event. In principle, all objects may proceed simultaneously. In practice, parallelism is limited by the number of processors available, as well as by other resources shared by all processes.

As each object computes, it updates its internal state to reflect the consequences of the event. This may involve a message dialog with the sender of the event. The result of the processing might be new events added to the priority queue, or it might simply update an object's internal state. When an object completes its processing, the object becomes free to process another event, if another one is waiting. When all of the objects have completed processing an event, the kernel deletes it.

4.3.2 Object Modules

Each object module is simply a C++ class compiled into VEA. An object of that class is instantiated only if it is specified in the configuration file. Objects can process events and messages simultaneously, in either a multi-tasking or a multi-processing sense. They can be thought of as asynchronous co-routines. This requires that objects have some safeguards that prevent more than one process from modifying common data simultaneously. VEA provides primitives, implemented with UNIX semaphores, that perform this function. This allows the programmer to lock critical portions of the code for read-access, or for write-access privileges to its data. Multiple processes are allowed to read data simultaneously, but only a single process may be granted write access at any one time.

4.3.3 Configuration Files

A configuration file holds the specifications for all objects and events to be instantiated and initialized at the beginning of a VEA session. The configuration file can be created and edited with a text editor. When a VEA session begins, it reads the configuration file, and sets the initial states of all the objects. When a VEA session ends, the final states of all objects and events are written to an output file, so that another VEA session can begin exactly where the previous one ended, if necessary. Since generic object modules are pre-compiled into VEA, a new session can be re-configured quickly by changing the configuration file.

4.4 FILTERS AND MANAGERS

To support device and application independence, VEA uses a layered approach [Grinstein 93]. The layers are called filters and managers.

The filters are objects that communicate with a specific device. Filters can perform device-specific optimizations, such as buffering data, removing unwanted sensor noise, or

using predictive filters (e.g., to compensate for system latency) on the data. The filters can encapsulate the data into an event, and post the event with the kernel, or pass the filtered data directly to an associated manager for further disposition. A primary purpose of the filters is to translate device-specific details into generic interactive modes. This encapsulation allows the system to grow and adapt as new devices are added.

Managers are objects that monitor a specific modality (e.g., gloves, text, graphics). They may apply higher-level optimizations, such as combining groups of related events into a single event.

For example, consider a DataGlove filter and a gesture manager. The filter reads the DataGlove serial port, collects finger angles, hand location, and orientation data. Periodically, it will post posture and locator events for the hand. These posture and locator events can be used directly by objects, such as a simulated hand graphic. However, the manager can also monitor these events. It can check for valid gestures (which we define as sequences of postures) and post gesture events as required.

As another example, consider a graphic filter and view manager. The graphic filter is primarily concerned with rendering the simulated scene as quickly as possible. The view manager, on the other hand, is concerned with monitoring locator events that are currently controlling the viewpoint within the system. In the case of a HMD, this would mean monitoring the locator for the head tracker, and updating the current viewpoint for the graphic filter. Alternatively, if the viewpoint is attached to a vehicle, it must track that vehicle's movement.

4.4.1 IRIS Performer

Since most interaction depends on visual display, the graphic output filter is a central software module. The primary tool we selected for graphic output is IRIS Performer, from Silicon Graphics. Performer is a software library built to support visual simulation applications. Performer has a number of features that were desirable for our development effort:

- full support for IRIS hardware capabilities (texturing, antialiasing, lighting)
- highly optimized for Silicon Graphics platforms
- supports multiprocessing parallelism
- compatible with Software System's MultiGen [SSI 92] modeling tools.
- low cost (less than $500)

There are several third-party vendors, however, who offer comparable tool kits, such as: Gemini GVS, Paradigm Simulation Vision Works, and Sense8 World Tool Kit. These products would be appropriate choices if platform portability is a major concern. We chose

IRIS Performer because we felt it would offer the best level of integration and performance on our Silicon Graphics workstation.

IRIS Performer directly supports multiprocessing. It will automatically establish separate processes for the application, scene culling, and drawing, as well as the appropriate data queuing mechanisms between these processes. This multiprocessing is in addition to the multiple processes used by VEA.

4.4.2 Graphics Manager

The graphic manager is central to the system, because nearly all user interactions utilize a visual interface. The VEA graphics interface uses IRIS Performer and IRIS Graphics Library (GL) to implement a graphics filter/manager. Due to the high degree of integration in the graphics hardware and library software, the VEA graphics module functions in several capacities as both a filter and a manager. This dual role is not entirely unique, but it does underscore the importance of this module.

4.4.2.1 Input Events

The graphics module acts as an input filter for several devices that are tightly coupled with the graphics hardware, such as the keyboard, mouse, dials, buttons, and SpaceBall. Due to multiprocessing restrictions, all device access through GL must occur in the drawing process. This data must then be fed back to the VEA interface process via shared memory. Input data are packaged into events of the appropriate modality, and passed to the VEA kernel for distribution.

4.4.2.2 Graphic Events

The graphics module acts as a manager for Graphic events. If the event contains the name of a valid graphic description file, this file is opened, read, and converted into the internal Performer graphic database format. The graphical representation is instantiated as part of the whole scene. The name of the originator of the event is attached to the graphical instance, so that it can be referenced later. Other kinds of events, such as "Highlight" and "UnHighlight," reference the object instance name to perform their operations. If an event requests the same file again, the manager will clone an instance from the representation already in the graphic database, so there is no need to read and convert the file again. This practice conserves both time and memory.

4.4.2.3 Posture Events

The graphics module also manages Posture events. A Posture event is emitted by a posture filter, such as the DataGlove interface module. When the manager reads in a graphic

description file, it can detect special comments inserted into the MultiGen Flight-format file by the modeler. These comments instruct how to interpret Posture events to control articulated objects. The comments specify which parameters control what parts, and the center of rotation and orientation of each articulated part. This information is stored in a table that links posture data to the appropriate graphic representation. As posture data is received, pointers in the posture table direct the data to the correct node in the scene database for modification. The data is picked up by the Performer processes on the next frame update.

4.4.2.4 Position Events

The graphic manager monitors Position events. If the names associated with the positions correspond to the names of any graphical representation, that position is immediately inserted into the scene database, so that the graphical representation tracks the simulated position. Furthermore, if the name of the current graphics viewpoint matches the name in the Position event, the viewpoint is modified correspondingly. Thus, Position events are used several different ways: to inform other simulated objects of changes in position, for collision detection, and for graphic update. Yet, the simulated object does not need to keep track of graphical positions at all. This is another example of how layering of abstractions can promote parallelism and extensibility.

4.4.2.5 Trajectory Events

A Trajectory event is a higher-level abstraction of the Position event. It specifies an object's path through space over time. This is accomplished by providing parametric coefficients for an object's initial position, velocity, and acceleration, with time as the parametric variable. The graphic manager maintains a table of object trajectories, and updates the position of every object in the table at the beginning of each frame cycle. The updated position of each object is also broadcast via a Position event, so that consistency is maintained throughout the simulation. This mechanism achieves smooth animated motion with minimal intervention from applications and simulated objects.

4.4.2.6 Text Events

The graphic manager also manages text input and display. As the user types on the keyboard, the characters are displayed. When the Enter key is pressed, the manager sends the typed string as a Text event. Application models may elect to subscribe to text events, and may interpret these events as direct commands.

For output, the graphic manager accepts Text events. If the name of the event originator matches that of a graphical representation, that graphical object is labeled with the text. The text tracks the object if it moves on the display screen. If the name does not match a graphical object, the text will be displayed in a scrolling text area.

4.4.3 Application Modules

So far, we have discussed filters and managers, which implement the basic functionality for a virtual environment. We still have to provide models and behaviors for the virtual world itself. The layering methodology introduced for devices applies in this case as well. The application can be decomposed into filters and managers, each of which provides a specialized service.

In many cases, it will be possible to encapsulate a computational model so that it can be integrated directly into VEA. For example, we have a class called Positional Object. This class subscribes to Locator and Position events, and generates new Position events. If a Locator event matches the locator name in a Positional Object, it uses that information to move its current location, and posts the new location in a Position event. All movement is relative to a base object. If a Position event matches the name of its base object, the Positional Object will track the base object, so that its position and orientation are always relative to the base object.

One might also want to integrate an existing piece of code, or a third-party application program. In this case, VEA provides an Application class. The Application class will setup an independent process for the application, so that it can run in parallel with the rest of the system. It also provides an event queue. The programmer needs to write only a simple interface, which passes event data to the application in the correct format, and conversely, posts application results as new events, as appropriate. The Application class takes care of queuing events, so that the application code can process each event at its own rate. This is the approach we took to integrate the knowledge base application, which is the subject of the next section.

SECTION 5

A KNOWLEDGE BASE APPLICATION

5.1 RATIONALE

VR represents a programmable human machine interface to an application. The Virtual Environment Architecture (VEA), described in Section 4, is an implementation of a VR interface that provides the necessary infrastructure for the integration of various VR devices, and an event mechanism so that high level device data can be sent to the application. The application, however, must *interpret* the device data. In order to provide a natural and intuitive interface to the application, the actions of the user must be interpreted with respect to the *domain* and the *context* in which the actions are performed. In addition, the application must be able to represent and manipulate complex, real-world domains. To do this effectively, we have incorporated the use of a *knowledge base* in VEA.

As mentioned in Section 3, one of the requirements of a VR system is the ability to integrate multiple input devices. The development of VR devices has increased the number of degrees of freedom generated, as well as introduced devices with new functionalities. VEA is capable of packaging the device data into categories that describe their functionality, or modality. However, the data must be interpreted by the application with respect to domain knowledge and the context. For example, if the user were to point to an object using a 3D selection device, and attempt to move the device in three space, the application would need to have some knowledge about the object that has been selected to determine the behavior of the object. In simple domains, we could interpret this to mean move the object to a new location. In more complex domains, object properties and functionalities may limit this sort of behavior; in a battle management application, it may be undesirable to move an aircraft. Knowledge about the object is needed to interpret the data provided by the selecting device. The result of using a 3D selection device to select a aircraft may differ from that of selecting a tank. For example, aircraft and tanks may have different constraints on their mobility. The ability to store this information, and to retrieve it when necessary, is accomplished by using a knowledge base.

Another requirement of the application is the ability to describe and simulate complex domains. We can represent and simulate an application either through a numerical simulator or a knowledge base. An application driven by numerical simulations uses mathematical models to determine future states of the system. For example, to simulate the movement of a tank, the current position, velocity and acceleration of the tank is used to calculate the position for the next visual frame. One of the drawbacks of this type of simulation is the response to queries such as "Why is the tank's speed 10 mph?" A mathematical simulator

would either not be able to understand the query, or respond with uninteresting information regarding position, velocity and acceleration. On the other hand, knowledge-based simulators infer facts about the possible future states of the system being simulated. Therefore, in the above example, a knowledge base would be able to respond with "Because the tank is on a road of type class 3 and when a tank is on this type of road, its maximum speed is 10 mph." For this reason, knowledge-based simulations can better explain the reasoning behind the results generated.

5.2 DEFINITIONS

In some sense, any computer program can be an "expert" at its task. However, some domains involve tasks that cannot be broken down into a simple set of mathematical calculations. In this situation, knowledge-based techniques provide assistance. Knowledge-based systems typically collect and store fragments of knowledge into a knowledge base, and then access and manipulate this knowledge base during reasoning about a particular problem; this is known as *inferencing*. Knowledge-based systems have had success in application areas ranging from diagnosis to planning and scheduling tasks.

There are four basic types of knowledge that need to be captured in a knowledge based system.

- *Object Knowledge* is based on classifying objects into classes that accurately describe the object's properties. Each object maintains a list of facts about itself. For example, the class Plane would store its maximum flying range and minimum take-off speed.

- *Event Knowledge* represents actions and events in the world and their cause and effect relationships. For example, the effect of an Identify Event for a plane could be to display information about itself. In the Object Oriented Paradigm (OOP), event knowledge is typically encoded as methods or member functions of a particular object class. The OOP has become one of the major representation schemes for representing object and event knowledge.

- *Performance Knowledge* involves knowledge about cognitive behaviors. This allows objects to know how to perform certain actions, like flying a plane.

- *Meta knowledge* is knowledge about knowledge. While object knowledge provides descriptions about a particular object, meta-knowledge provides descriptions about the object's type. This allows for reasoning capabilities on sets of objects. For example, since we know that aircraft can fly, then we can interpret the Text Event "mobilize" to mean take off. In the case of a Tank, this same event

would be interpreted to mean move forward. Without meta-knowledge, this information would need to be stored explicitly as object knowledge.

The method by which these types of knowledge are stored is its *knowledge representation*.. Although there are many techniques for knowledge representation, all fundamentally structure the knowledge of the problem domain in a declarative manner. This means that the facts and relationships in the problem domain are encoded explicitly, allowing other programs to use them in their reasoning processes. This contrasts to more standard computation that relies on programs and subroutines to implicitly create new facts and implement relations between them. The addition of new facts or relations in these cases requires the re-coding of the program; whereas, the addition of new facts or relations in a knowledge-based system requires only their addition to the knowledge base. Object-orientation (i.e., that which we find in C++) is a form of knowledge representation. However, a simple object-oriented data-structure does not guarantee knowledge-based capability. For example, while an encyclopedia is a source of knowledge, a reader is needed to understand and reason about this knowledge [Barr 81]. Therefore, explicit rules or relations for reasoning as well as a mechanism for their manipulation are required.

The knowledge base application for VEA uses a hybrid approach of concept hierarchies and a production system for its knowledge representation. A concept hierarchy is a structure of classes, based on the OOP, that allows for encapsulation of object knowledge and behavior. The classes in the concept hierarchy range from abstract concepts such as *Virtual Objects* to more concrete objects such as *F15* . The classes are arranged in a tree structure to allow for inheritance of object properties and behaviors. So, for example, the class *F15* would inherit attributes and behaviors from the class *Plane*.

We also include a production system, often refereed to as a rule based system, to allow for additional capabilities. A production system is composed of a set of facts and rules. Facts represent information about the world. For example, if a tank were on a road class of type 3, then we could add the following fact to our knowledge base:

road_type(tank1,ROAD_CLASS_3)

Rules are stored in the form of condition-action pairs. For example, a rule for tank mobility would look like:

If road_type(tank1,ROAD_CLASS_3) **Then** maximum_speed(tank,10)

When the condition part of the rule is satisfied (i.e., the fact or facts are in the knowledge base), then the rule is activated or fired and the action part of the rule is executed. In the above example, the fact

maximum_speed(tank1,10)

would be added to our knowledge base. Rules are fired by the underlying inference engine. When new facts are added to the knowledge base, the inference engine determines if there are any new rules that can be fired (i.e. the condition part of the rule is satisfied).

The are several advantages of using a hybrid approach. First, it allows us to create two separate knowledge bases, one for the basic behaviors which are needed to interact with VEA, and one to represent the domain. This is accomplished by creating two concept hierarchies, one for domain independent information, called the core hierarchy, and one for domain dependent information that inherits functionality from the core hierarchy. This proves beneficial when new applications and domains are included; only the domain dependent hierarchy needs to be implemented. Secondly, by including the concept hierarchies in the production system, we can store much of the knowledge in the class model, reducing the number of facts needed by the knowledge base. In turn, fewer rules will be executed. This will allow for a greater amount of knowledge to be stored with greater efficiency.

5.3 REQUIREMENTS

There were two absolute requirements for the representation of the VR knowledge base: that it conform to the OOP, and that it interface to VEA. However, in satisfying each of these requirements, there were a wide range of desirable, and sometimes disjoint, features that could aid in satisfying the particular requirement, or could add additional behavior. The two requirements and their possible features are discussed below.

The OOP was selected as the unifying theme of the VR work; VEA was designed and implemented in the OOP. We continued the OOP in the design and implementation of the Knowledge Base. However, there are additional features that were needed for the implementation of the knowledge base. The first is a requirement for inheritance of attributes and behaviors both statically and at run-time. Secondly, dynamic type checking and error checking provided the ability to specialize behavior based on the object's type. In addition, advanced behavior combination provides more flexibility and control of behavior. These features provide the ability to capture Performance and Meta knowledge which is typically missing from C++.

Since VEA was written in C++, the knowledge base must provide a flexible interface to the C++ /C language. The communications protocol between VEA and the knowledge base must provide for high bandwidth message passing. This is crucial since it was difficult to determine the level and frequency of communication required between the kernel and knowledge base.

Additional features had a positive impact on the implementation of the knowledge base. Since the knowledge base used a production system, we did not want to write our own inference engine. Other features/issues considered were portability, cost, maintainability, longevity, and ease of use.

5.4 SELECTION OF KNOWLEDGE BASE TOOL

The primary options considered for implementing the knowledge representation format included: 1) Designing a custom representation format in an existing language; or 2) Integrating a C-based expert system development shell. To investigate these options, we considered the tasks and difficulties involved in using an existing language, and we conducted surveys of available language extensions and expert system shells. The results of the activity are described in the following paragraphs.

Both of the absolute requirements can be met by implementing the knowledge base directly in C++. This approach would best satisfy the integration requirement, and provide for the greatest performance. However, due to the lack of behavior combination, run-time inheritance, and dynamic type checking, many of the standard conventions required for implementing knowledge bases would be unsupported. In addition, development time would increase dramatically and the implementation would not be straightforward. Lisp has been used extensively throughout the Artificial Intelligence community for knowledge representations and would provide with the greatest flexibility. However, a number of integration problems arise. First, it is very difficult to call Lisp functions from external applications. Another problem is where the Lisp code would reside. If it were to run on a separate workstation, inter-machine communication problems arise. These two languages therefore fall at either end of a spectrum. Lisp fully satisfies the representation requirements, but lacks in the interfacing capabilities. On the other hand, C++ is a natural selection for interfacing to the kernel, but lacks the representation powers of LISP.

A variety of C-based expert system shells provide object-oriented knowledge representation schemes that could have potentially be integrated into the kernel and provide a more flexible knowledge representation alternative to C++. Generally, these knowledge representation formats tend to lie in the middle of the expressiveness/ease of integration spectrum; they are not as expressive as Lisp but, since they compile into C, can be more readily called from within the kernel and can run very efficiently. A listing of fully-featured C-based expert system shells is shown in table 5.

Table 5. Available Expert System Shells

Tool Name	*Vendor*	*Platforms*	*Remarks*
Automated Reasoning Tool for Information Management (ART-IM)	Inference Corporation El Segundo, CA 800-322-5590	PCs ($8,000), workstations ($12,500), mainframes ($150,000). Not available on SGI workstations.	Efficient forward chaining rule interpreter, Lisp-like syntax, object-oriented extensions now in Beta release.
C-Language Integrated Production System (CLIPS)	Supported by Computer Sciences Corporation for NASA/JSC. 713-280-2233	Source code available free to government contractors; can be compiled on PC, MAC, UNIX, and VMS workstations.	Similar syntax to ART-IM and efficient forward chaining rule interpreter, includes object-oriented programming language.
Knowledge Engineering System (KES)	Software Architecture and Engineering, Inc. Herndon, VA 703-318-1000	PCs ($4,000) workstations ($10,000) minicomputers ($25,000) mainframes ($60,000)	Supports backward chaining and hypothesize and test inference strategy.
Nexpert Object	Neuron Data Palo Alto, CA 800-876-4900	Macintosh and PCs($5,000) UNIX workstations, including SGI ($12,500) minicomputers mainframes	Superior development interface and backward/ forward chaining rule interpreter. Comprehensive object extensions in Beta release.
ProKappa	IntelliCorp, Inc. Mountain View, CA 415-965-5700	workstations ($10,000) PC-based tool available but not fully compatible. Not available on SGI.	Comprehensive object-oriented programming environment and forward/ backward rule interpreter.

Of the tools, the C-Language Integrated Production Systems (CLIPS) provides for a intuitive interface to the C language, and is also capable of representing the four required types of knowledge [Giarratano 91]. It was developed under NASA/Johnson Space Center during the mid-1980s and is now supported by Computer Sciences Corporation under NASA funding. It provides a Lisp-like syntax, a fully-featured object-oriented programming extension with method combination and daemons, and an extensive Application Programmer's Interface (API). It also incorporates a forward chaining rule interpreter based on the Rete algorithm that is optimized for efficient run-time execution. In addition, it provides a powerful window interface that aids in the implementation and debugging of the system. Extensions to this interface may constitute a Command Line Interface. Since the C source code is provided, CLIPS can be compiled by users onto the platform of choice.

5.5 INTEGRATION OF CLIPS

5.5.1 Overview

Figure 5.1 shows the event mechanism in VEA. In integrating CLIPS, we constructed a manager in VEA that serves as a link to the knowledge base. This manager, written in C++, is responsible for receiving the events from VEA and sending them to the knowledge base. Any display or feedback data that is generated by the knowledge base is also handled by this manager. It also serves as a translator between the two languages C++ and Clips. The manager is also responsible for interprocess communication since the knowledge base runs on an independent process.

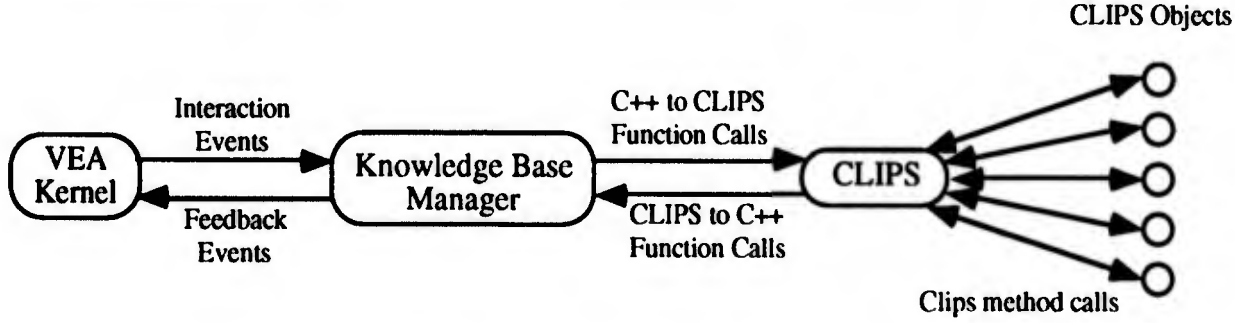

Figure 5.1. Integration of Clips and VEA

5.5.2 Hierarchies

A concept hierarchy is used as our main representation scheme, and where appropriate, rules and constraints are also used for additional functionality. We compose the knowledge base in two parts. The first part is a domain independent concept hierarchy that performs basic operations such as receiving events, and provides default or basic characteristics and behaviors. An application dependent concept hierarchy is also constructed. It is through this hierarchy that we are able to interpret the user generated events based on the domain.

5.5.3 Domain Independent Hierarchy

The domain independent hierarchy, or core, describes the basic features of the knowledge base. We include three types of classes or trees: Top Level object classes, Generic Slot Classes, and Generic Mixins.

The top-level classes consist of *Vr Object*, *Environment*, and *Domain Object*.. The *Vr Object* class represents the root of the of all Virtual Objects. We call this tree the Object Hierarchy. These objects register their interest in events of a particular type through the subscription mechanism in VEA. Although it may appear that all objects should receive all events, there are certain concepts or objects where this would not make sense. For example, an Operation or Mission would not benefit from user data provided by the Locator event. In some applications, objects may add or drop subscriptions dynamically. This behavior is also included in the *Vr Object* class. Inheriting behavior from *Vr Object* are the *Environment* and *Domain Object* classes. The *Environment* class is included to act as an object manager and to serve as a link to the VEA knowledge base manager. This class receives events from the VEA knowledge base manager, and distributes the events to the appropriate objects. If no objects subscribe to a particular event type, then the environment will inform VEA not to send those type of events. The *Domain Object* class serves as the basis for the domain object hierarchy. This class is used to set up the mechanism for responding to events. We include a method for responding to particular event types for each descendant of *Domain Object*. By layering these methods, it becomes possible for an object and all of its ancestors to determine the behavior for a given event type.

We also implement Generic Slot and Mixin classes. These classes will work to provide default characteristics and behaviors to the core hierarchy. The Slot Class structure implements characteristics that a variety of classes require. For example, we include the slot class *Position* to describe a point in n-space. This class has all of the basic functionality for moving the point in the Cartesian coordinate system. While the Slot Classes are instantiated for use by the objects in the virtual world, they are not used for inheritance. This falls on the duties of the Generic Mixin classes. These abstract classes, which are never instantiated directly, are used to provide additional behavior to the domain classes. For example, we include a class called *Positional Object* for any object that can have a position. One of the default instance variables of this class is an instance of the Slot Class *Position*. The class *Positional Object* has default behaviors such as responding to Locator Events.

5.5.4 Domain Dependent Hierarchy

It is through the application-dependent concept hierarchy that we are able to interpret the events with respect to the domain. Our domain is a battle commander's assistant. One of the key features of a virtual environment system is its ability to enhance the situational awareness of the user. This comes about, in part, by enhanced access to information relevant to the decision making process. The application focuses on supporting the actual tasks a battle commander must perform. It provides for environmental query, information retrieval and asset/resource allocation directed at single objects. This application requires the ability to select and query objects in the virtual world about their current state, and to manipulate these objects to perform a specific task.

The domain dependent knowledge base is constructed in the same manner as the core hierarchy. Particular detail is taken to model tangible and intangible abstract concepts and behaviors. This includes such things as Planes and Tanks, as well as Missions and Operations. At the bottom of the hierarchy are those objects that will be instantiated to make up the Virtual World. For each class in the hierarchy, we construct event handlers that are specific to a particular type of event. Therefore, one class may have several event handlers if it accepts several types of events. This will allow for objects to respond to events based on their current state. As an example, the subtree for the class *Aircraft*, along with its ancestors in the core representation is shown in figure 5.2.

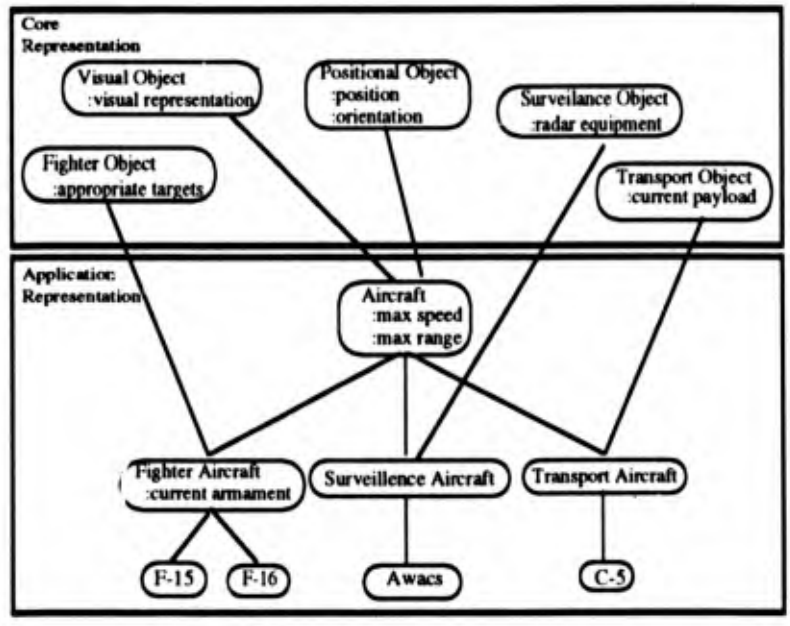

Figure 5.2. The Aircraft Hierarchy

The scenario calls for the ability to retrieve information of an object. The user issues the command "identify object," either through spoken word or a keyboard, and text about the object is displayed. This text display is based on the object's type and current state. For example, an C-5 would display is current payload, while an F-15 would display its armament configuration. In each case, the class determines the text to be displayed.

Another determining factor of what text the object displays is the context in which the user poses the question. We include in our domain knowledge base a focus model. This model allows the knowledge base to keep track of the user's current task and interest. Therefore, if the user was in the process of planning an Offensive Counter Attack, and directed the command "Identify Object" to an F-15, then the aircraft would also include information about targets appropriate for that aircraft. A more detailed description of the prototype demonstration of a battle management scenario is provided in section 6.

SECTION 6

PROTOTYPE DEMONSTRATION

This section discusses our prototype demonstration application, a "battle managers assistant." We describe the hardware configuration, how the prototype application works, and how well it performed.

6.1 HARDWARE CONFIGURATION

For our demonstration, we selected devices that we felt were most appropriate for use by military commanders. In particular, we believe that any device encumbered by tethers would be unacceptable for this type of application. In addition, we wanted to demonstrate this prototype to groups of people. Accordingly, we did not emphasize the use of head-mounted displays and gloves. However, we feel these devices would be appropriate for other types of military applications.

The center of the system is a Silicon Graphics IRIS 4D/340 RealityEngine. This is a general-purpose, UNIX-based workstation, with very good graphics capabilities. It contains four 33-MHz MIPS R3000 processors, each of which is rated at about 20 SPECint92 (a CPU benchmark index). The RealityEngine refers to the graphics subsystem, which supports up to 600,000 textured polygons per second. Our configuration has two Raster Manager boards, and a Multi-Channel Option board.

This configuration allows us to output two separate channels, each at a resolution of 1280×1024 pixels. The two channels are used to drive stereoscopic displays. A pair of video distribution amplifiers are used to drive both a n-Vision Quarterwave HMD and our large screen display. Our large screen display consists of two high-resolution Electrohome CRT projection units, with linearly polarized sheets mounted over the projection lenses. Complementary polarizing glasses, available from standard suppliers, are worn by the viewers. The large screen display is used almost exclusively; however, one user may wear the HMD, while others observe on the projection display.

We use a Logitech 3D mouse (acoustic tracking) to control a virtual "wand" (see appendix section A.1). The wand is used to designate objects, by pointing and pressing the mouse button. The mouse is tethered, but the user is able to put down the mouse at any time. Optionally, an Origin Instruments DynaSight (infra-red optical) tracker is used to track the principal viewer's position (section A.2). A reflective dot placed on the user's polarizing glasses achieves tetherless head tracking. This provides the user with a "look-around" capability, with the view automatically adjusting to the user's position in the room.

A Macintosh SE personal computer, attached to the IRIS via RS-232 serial port, is equipped with an Articulate Systems Voice Navigator. The Voice Navigator can be trained to recognize single words or short phrases. This unit allows the user to control the application by spoken command.

6.2 DESCRIPTION

A prototype demonstration was developed using VR as a tool to support crisis operation at the unit level. The tool allows an airbase Unit Commander to access various types of information sources that are need during the planning stage of a mission. The value added of a VR interface is threefold:

- It provides a greater sense of situational assessment.
- It allows for an intuitive interface to multiple types of information.
- The use of advanced natural interaction techniques allows for a greater coupling between the user and application.

The particular scenario is for supporting a relief mission to the former Yugoslavia. The relief mission entails a C-5 flying supplies into an airbase located near hostile threats. The unit commander of the airbase must secure the airbase from potential threats to support the mission. To accomplish this, there are three basic steps:

- Identify potential threats
- Determine aircraft vulnerability
- Provide an air escort

The current method for solving the three basic steps involves gathering information from multiple sources, and using several unconnected tools. To identify enemy assets within a given range of the airbase, the Unit Commander must gather information from intelligence reports which may either electronic, or written text. The potential threats are then identified by the type and positioned on a contour map. To determine vulnerability, the unit commander uses intelligence information about the threat, and possible line of sight information that is gathered from the contour map. Lastly, any planning for neutralizing the threats and providing air escorts to the C-5 must be done using a planning system.

TheVR tool allows all of these tasks to be combined into one space, and provides greater situational assessment and natural interactions. The scenario begins with a view of the airbase along with the objects that are currently located at the airbase (figure 6.1). These objects are represented as objects in the knowledge base and are displayed using textured graphics in Performer. The user first speaks "identify threats" using the Voice Navigator. The names of the potential threats are displayed as a text list on the screen. In addition, the

insert figure 6.1 name or names of the primary threats are also displayed. In our particular scenario, there are two potential threats, an M1 tank and an SA-7. Rules in the knowledge base determine that the SA-7 is the primary threat.

The user then says "show 2nd threat." Since the threats were displayed using a list, the knowledge base interprets this to mean show the second item in the list, which in our case is the SA-7. The user's viewpoint is changed so that the SA-7 appears on the screen. Rather than simply changing the coordinates of the viewpoint, we animate the movement from the current viewpoint location to the best location to view the object. This provides the user with a better sense of the object's location. The user the says "identify it" and information about the SA-7 appears next to the object. The knowledge base is capable of understanding anaphoric references so it knows the user is referring to the SA-7. In addition, the knowledge base determines what text is appropriate to display based on the object's type and the context in which the query was posed. In this example, the SA-7 only displays its affiliation (figure 6.2). To determine if the SA-7 is a threat, the user must see the SA-7's line of sight by speaking "show viewpoint." The knowledge base attaches the viewpoint to the most recent object in the user's focus. The viewpoint is moved to display what the SA-7 can see (figure 6.3). This allows the user to determine that the SA-7 is not a threat.

To view the first potential threat, which is the M1 tank, the user says "show 1st threat." Again the viewpoint is dynamically moved to show the M1 tank. The user says "identify it" to retrieve information about the object (figure 6.4). The user determines that, given the tank's range, it presents a threat to the success of the mission. The user will plan a task to neutralize the enemy threat by speaking "plan ground task." The knowledge base determines that since the enemy tank is in the user's current focus, it will become the target for the ground task. The user will then need to assign assets to carry out the ground task. To identify the assets that are relevant to this particular task and that are available, the user speaks "identify assets." The names of the objects are displayed in a list as text. The user can view the assets by speaking "show them." The user uses the logitech mouse to select the assets to use for the mission and speaks "use those." The assets selected are then used for the ground task. To view the ground task, the user says "identify it" (figure 6.5).

The last step of the operation is providing an air escort for the C-5. The user will speak "plan air task" to create the task. To identify the potential assets, the user speaks "identify assets" and a list of two F-16s appear. The user can use both of these by speaking "use those." To simulate the plan, the user speaks "execute." In this particular case, the aircraft begin to takeoff, after receiving clearance from the control tower (figure 6.6).

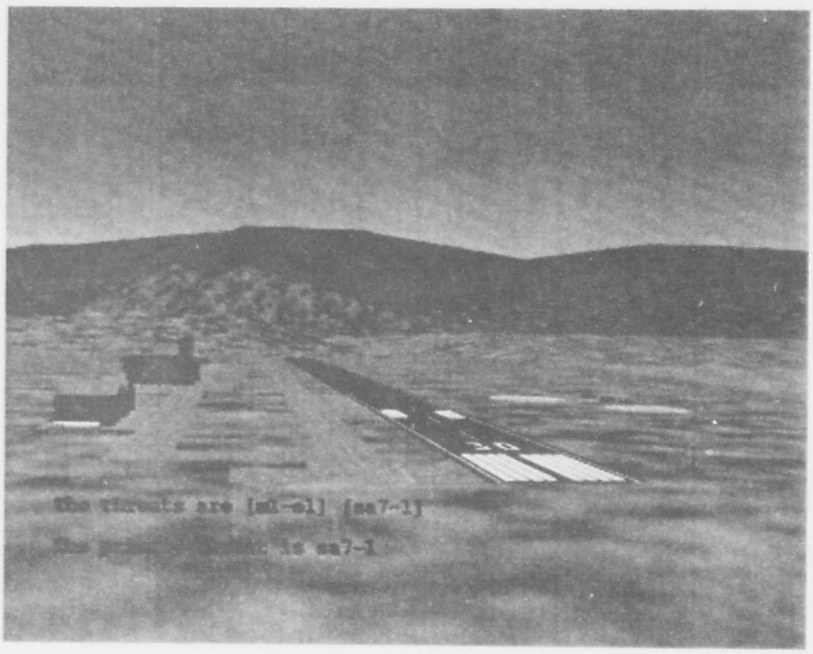

Figure 6.1. View of the airbase.

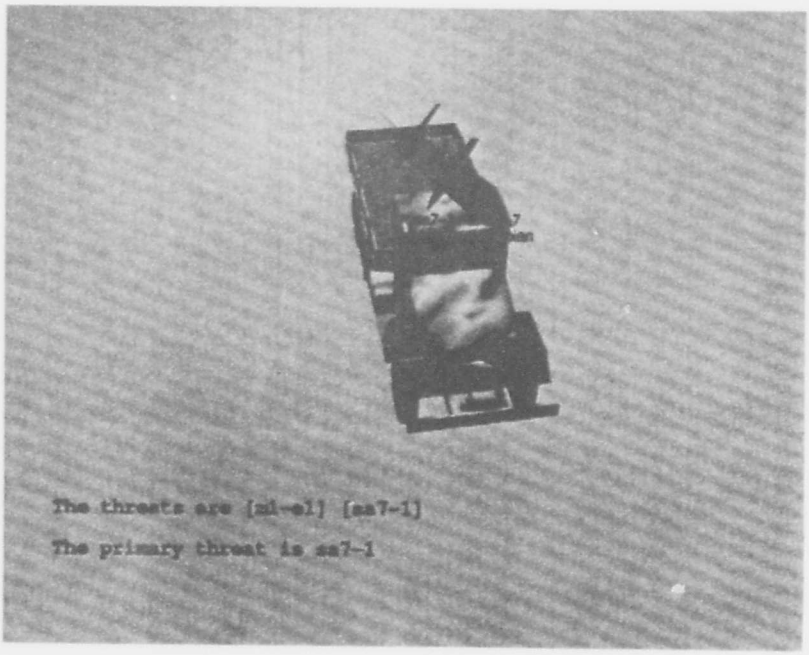

Figure 6.2. Surface-to-air missile launcher.

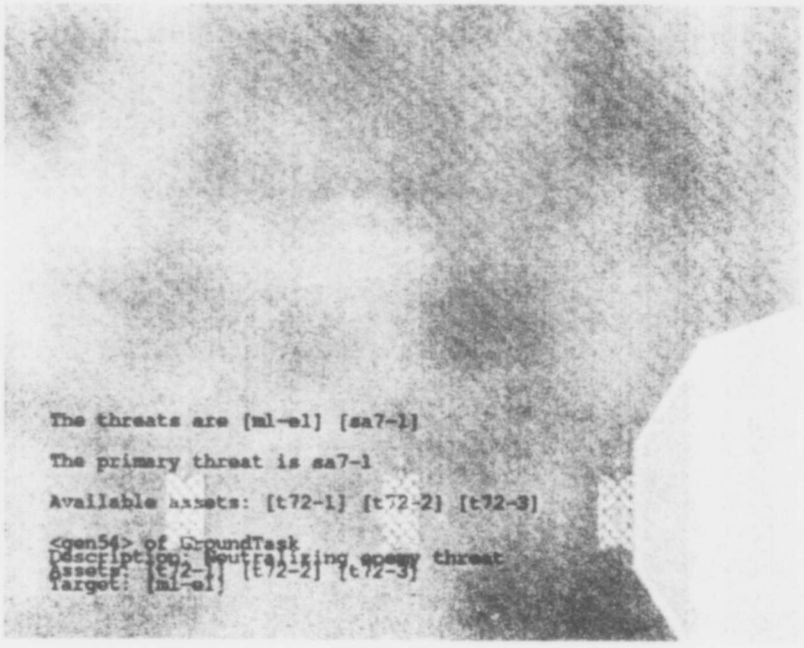

Figure 6.5. Assets for ground task.

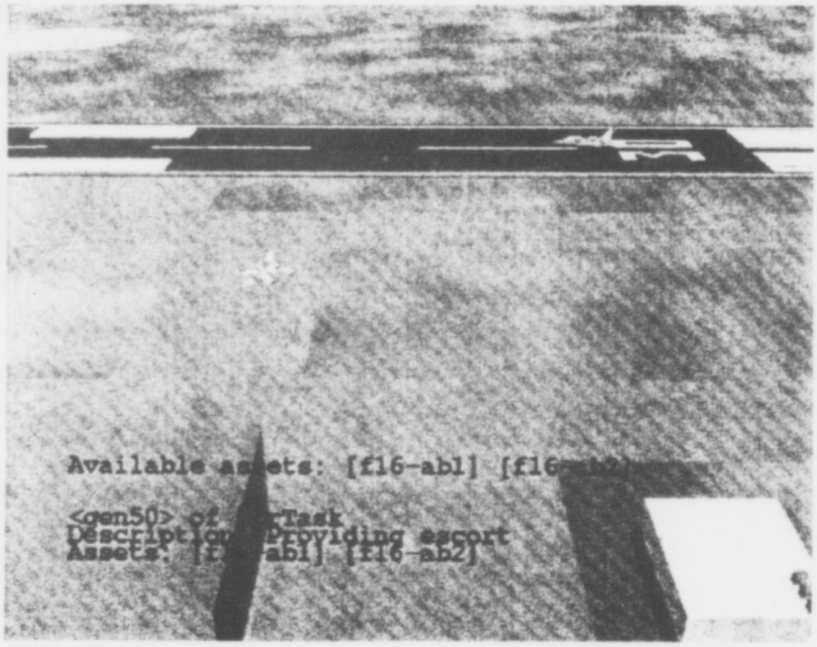

Figure 6.6. Escort aircraft take-off.

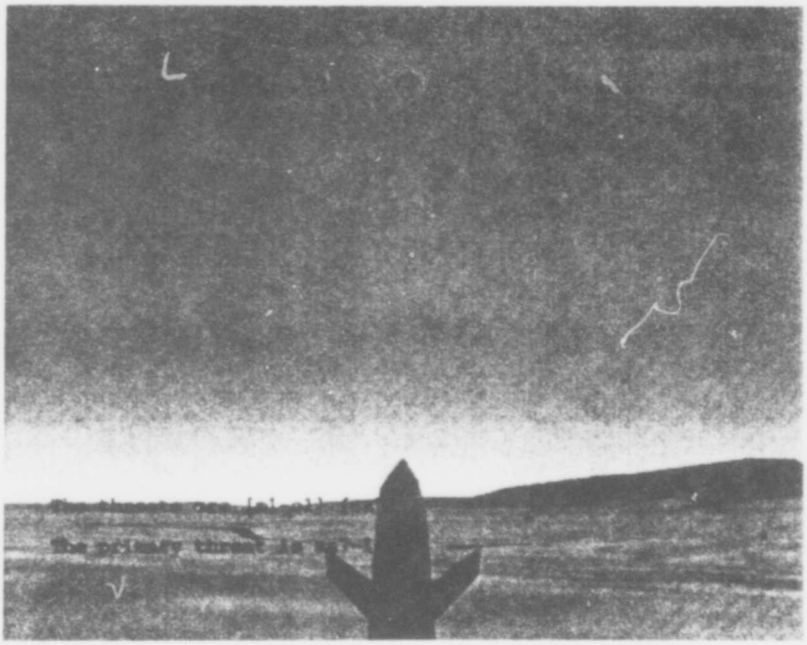

Figure 6.3. View from missile launcher.

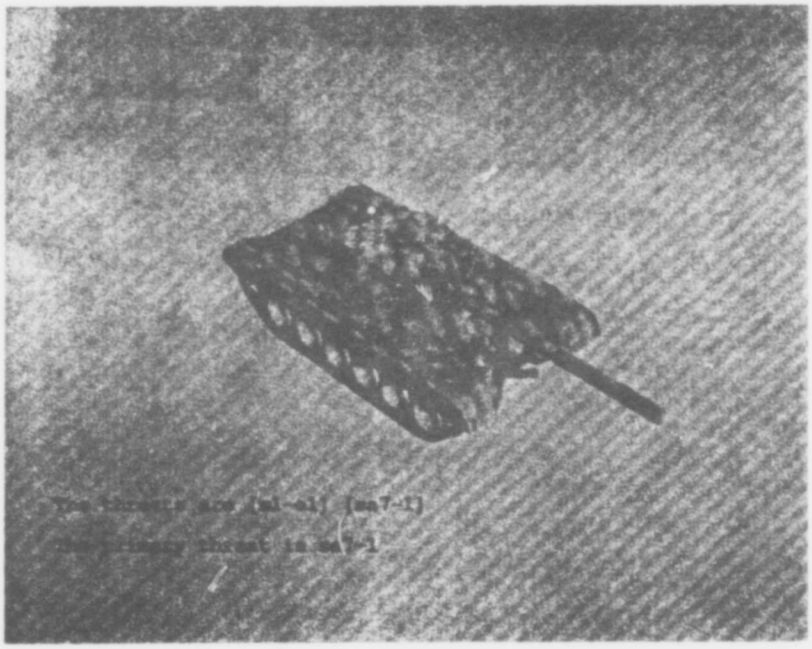

Figure 6.4. View of the tank.

6.3 PERFORMANCE

In this application we were able to sustain interactive frame rate of 10 to 20 frame updates/sec, with a fully textured terrain scene containing multiple dynamic moving models. The scene's total polygon count varied depending on the number and complexity of specific scene elements. The knowledge base was able to sustain 130 transactions per second.

Our experience with this application indicates that careful modeling, taking full advantage of level-of-detail switching, is extremely important for consistently high frame update rates. It is very important, for best performance, to have a least two levels of detail for each object model. A low-resolution model, which may be nothing more than a box, is actually required much more often than a fully detailed, high-resolution model. This is because, in a perspective view, most object models will be viewed from a distance. Furthermore, since the field-of-view is a frustum (truncated pyramid) shape, there are many objects in the background, and only a few in the foreground. Finally, even the high-resolution models should be kept as simple as possible. To a large extent, the simplicity of a geometric model can be compensated by using textures to supply the details.

There is a significant difference in models required for real-time visual simulation, versus models required for still (possibly ray-traced) renderings. For example, we purchased many of our military vehicle models from Viewpoint Inc. In several cases, even their lowest-resolution models contained 500-1000 polygons. When texture-mapped, even these "low-resolution" models look quite good. However, in the actual application, a model is seen close-up only seldomly, so these models serve mostly only to slow down the frame update rate.

We used the MultiGen modeling tool, from Software Systems Inc. (SSI), for virtually all of our modeling. MultiGen is full-featured, but it proved to be a bottleneck for scenario generation. Modeling can be a complex job, in which the expertise and judgment of the designer can have a tremendous impact on system performance. The MultiGen Flight format file to IRIS Performer conversion utility, supplied by SSI, did not support some features. As a result, terrain models based on DMA terrain and culture data could not be prepared in time for our demonstration. SSI reports that these discrepancies should be fixed in their MultiGen Version 14 software release.

SECTION 7

SUMMARY AND CONCLUSION

7.1 POTENTIAL ENHANCEMENTS

Our view is that potential applications outclass by far the state-of-the-art in virtual environments. The immediate challenge is to construct an infrastructure that supports a wide variety of virtual environment investigations. VEA does this by taking a modular, object-oriented programming approach. Input and output devices are embedded in filter and manager layers, which encourages device and application independence, and flexibility. Simulation and modeling are supported through the integral real-time clock, and the knowledge base. Rapid prototyping is supported through the use of flexible configuration files, which allow each VEA session to be tailored for a different use. High performance is obtained with built-in support for concurrent multiprocessing.

We have a number of areas in mind for expansion. Our immediate plans are to expand from a single-platform, multiprocessing system, to a networked, multi-platform, collaborative system for multiple users. Virtual environments represent a technology by which remote teams can work together on science and engineering problems. Following that, we envision adding intelligent agents. These agents could monitor the environment, and notify the user when certain situations arise, or carry out tasks on behalf of the user.

We would like to include support classes for physically based modeling. This would include basic Newtonian dynamics and collision models. This would allow us to create many interesting Virtual environments, in which objects behave as one would expect. Many object behaviors are completely mechanical, so there is no need for the knowledge base to intervene in these cases.

Another potential area includes advanced artificial intelligence information systems. Many applications exist for advanced user interfaces for schedulers, and reactive and adversarial planners. Virtual environments can provide an appropriate interactive environment for these applications.

7.2 POTENTIAL APPLICATIONS

Applications for Virtual environment technology are real, and are here today [Breen 92a, 92b]. However, there is much engineering work needed to realize the potential. We have already discussed military applications. There are many nonmilitary applications, as well.

7.2.1 Power Utilities

MITRE is actively participating in the development of a simulated control panel for a fossil-fueled power plant control room. Power utility companies throughout the country must train and rehearse power plant operators in correct procedures. Currently, mockup control room trainers are constructed full-scale from the actual devices. Such trainers are very expensive to build, operate, and maintain. Virtual environments are potentially a low-cost alternative that can be tailored to the needs of individual power plants. In this application, integration with a proven numerical simulation is critical.

Another application targets training for high-voltage switch and transformer yard repair procedures. Severe accidents, resulting in deaths, are currently a reality for many power utilities. Training and refresher courses in proper safety procedures are essential to reduce these occurrences. Virtual environments may offer a way to perform such training effectively, without exposing the operator to hazardous conditions.

7.2.2 Health

A recent conference focused on how VET could help persons with various disabilities [CSUN 92]. For example, a person with Parkinson's disease often has an involuntary shaking in the hands, but is unaware of this motion, unless he looks at his hands. A glove device, however, can be programmed to filter out the involuntary motion, and present a stable representation of the hand. Thus, the presentation of the hand would match the user's mental image. This could enable such persons to perform normal activities in the virtual environment, that would be difficult to perform in a real environment.

LIST OF REFERENCES

Appino, P. A., J. B. Lewis, L. Koved, D. T. Ling, D. A. Rabenhorst, C. F. Codella (1992), "An Architecture for Virtual Worlds," *Presence: Teleoperators and Virtual Environments* 1(1).

Ascension, ATC, "The Flock of Birds Installation and Operation Guide," Ascension Technology Corp., Burlington, VT, December, 1992.

Barr, A., E.A. Feigenbaum, "The Handbook of Artificial Intelligence", Vol. I, p.143-152, Heurishtech Press, Stanford, CA 1981.

Blanchard, C., and A. Lasko-Harvill, "Humans, the Big Problem in VR," *Implementation of Immersive Virtual Environments,* ACM Siggraph Seminar Course Notes, 1992.

Breen, P. T. (1992), *Near-Term Applications for Virtual Environment Technology*, M92B0000011, The MITRE Corp., Bedford, MA.

Breen, P. T. (1992), "The Reality of Fantasy: Real Applications for Virtual Environments," *Information Display* 8(11) 15-18.

Bryson, S., "Measurement and Calibration of Static Distortion of Position Data from 3D Trackers,"*Implementation of Immersive Virtual Environments, Course Notes 9*, Siggraph, 1992.

Bryson, S., S.S. Fisher, "Defining, Modeling, and Measuring System Lag in Virtual Environments," *Implementation of Immersive Virtual Environments, Course Notes 9*, Siggraph, 1992.

Clapp, R. E. 1987. "Stereoscopic Perception," *SPIE Proceedings*, Vol. 761, pp. 79-84.

Coco, G., *VEOS 2.0 Tool Builders Manual*, Human Interface Technology Laboratory, University of Washington, May, 1992.

CSUN, *Proc. Virtual Reality and Persons with Disabilities Conf.*, California State University Northridge, 18-21 March 1992, Los Angeles, CA.

Dunn-Roberts, R., K. Uliano. 1992. P. Moskal, and M. Moshell, "Head Tracking and Head Mounted Displays for Training Simulation," Final Report, Visual Systems Laboratory Document VSLM92.4, Institute for Simulation and Training, University of Central Florida, IST-TR-92-12.

Ellis, S. R., "Nature and Origins of Virtual Environments: A Bibliographical Essay," *Computing Systems in Engineering*, Vol. 2, No. 4, pp. 321-347.

Giarratano, J. C., "CLIPS User's Guide," Software Technology Branch, L. B. Johnson Space Center NASA, 1991

Grinstein, G. G., R. B. Mitchell, D. A. Southard (1993), "Virtual Reality: An Interface Architecture for Interactive Simulations," *Proc. Summer Computer Simulation Conf.*, 19-21 July. Boston MA, 259-264.

Hill, R., "Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction: The Sassafras UIMS," *ACM Transactions on Graphics*, Vol. 3, pp. 179-210.

Hughes Display Products Corporation. 1992. "CRT Assembly Specification Sheets: Models MH 1425, MH 1426, MH 1380, MH 1424, H 1444, 3H 1420, 3H 1442," Lexington, KY.

LEEP Systems, Inc., "LEEP Standard Product Guide with Domestic Prices," Product Announcement, 241 Crescent Street, Waltham, MA, undated.

Lipton, L. 1982. *Foundations of the Stereoscopic Cinema*, New York: Van Nostrand Rheinhold Company.

Logitech, "2D/6D Mouse Technical Reference Manual," Logitech Inc., Fremont, CA, 1991.

Masterman, H. C., G. G. Grinstein (1993), "Software Requirements for Virtual-Environment Applications," *SID Digest* **24**, 819-822.

Meyer, K., H.L. Applewhite, F.A. Biocca, "A Survey of Position Trackers," *Presence*, Vol. 1, No. 2, pp. 173 - 200, Massachusetts Institute of Technology, Spring, 1992.

Mitchell, R. B., J. L. Segal (1992), "A Virtual Maintenance Trainer," *SID Digest* **23** 906-908.

OIC, "The DynaSight Sensor Developer Manual," Version 1.0, Origin Instruments Corp., Grand Prairie, TX, 1993

OIC, "The DynaSight Sensor User Manual," Version 1.0a, Origin Instruments Corp., Grand Prairie, TX, 1993

Polhemus, 3Space™ User's Manual, Polhemus/Kaiser Aerospace and Electronics Co., Colchester, VT, May 1987

Robinett, W., and J. P. Rolland. 1991. "A Computational Model for the Stereoscopic Optics of a Head-Mounted Display," *SPIE Proceedings*, Vol. 1457, pp. 140-160.

Segal, J. L., W. J. Collins, R. B. Mitchell (1991), *A Prototype Virtual Training System*, M91-11, The MITRE Corp., Bedford, MA.

Sharp Corporation. 1991. "Color TFT - LCD Module," Product Bulletin.

Southard, D. A. 1992. "Transformations for Stereoscopic Visual Simulation," *Computer and Graphics*, Vol. 16, No. 4, pp. 401-410.

Southard, D. A., (1992), "Transformations for Stereoscopic Visual Simulation," *Computers & Graphics* **16**(4) 401-410.

Southard, D. A., (1991), "Terrain Visualization with Superworkstation Graphics," *Proc. SID* **32**(1) 39-47.

Southard, D. A., (1991), "Piecewise Planar Surface Models from Sampled Data," in N. M. Patrikalakis (ed.), *Scientific Visualization of Physical Phenomena*, Springer-Verlag, Tokyo, pp. 667-680.

Southard, D. A., (1991), *Superworkstations for Terrain Visualization: A Prospectus*, RL-TR-91-105, Rome Laboratory, Griffiss AFB, NY.

SSI (1992), "MultiGen User's Guide," Software Systems Inc., San Rafael CA.

Sturman, D. (1992) *Whole-Hand Input*, Ph.D. dissertation, Media Arts & Sciences, Massachusetts Institute of Technology, Cambridge, MA.

Taber's Cyclopedic Medical Dictionary, F. A. Davis, Philadelphia, PA, 1989.

APPENDIX A

DEVICE SPECIFIC MEASUREMENTS

The devices we evaluated represented a significant sample of the most viable products offered on the market at the time this evaluation began. This study is intended to provide a basis for evaluation of other devices which were unknown to us or otherwise unavailable at the time this effort began.

The following absolute position devices were evaluated:

- Logitech 2D/6D Mouse
- DynaSight Sensor
- Ascension Technology Flock of Birds
- Polhemus Isotrack

The Isotrack and the Flock of Birds are the most frequently used position trackers because they do not have blind spots caused by orientation.

A.1 LOGITECH 2D/6D MOUSE

The Logitech 2D/6D Mouse, manufactured by Logitech Inc. of Fremont, CA, is shown in figure A.1. It uses an ultrasonic technique to determine position/orientation. The device can operate either as a conventional 2D mouse or as a full six dimensional mouse. In the VR context, we are primarily interested in the six-dimensional application. The six dimensions are *x*, *y*, *z*, *pitch*, *yaw* and *roll*.

A.1.1 Components

The system comprises an external power supply, a triangle, a mouse, and a control unit. The triangle contains three ultrasonic transmitters, and the mouse is equipped with three ultrasonic receivers. The mouse conveys position data to the control unit, and the control unit calculates the position and orientation of the mouse and returns it to the host computer. The mouse is also equipped with four buttons, the status of which is also sent to the control unit on each reading. The control unit communicates serially with the host computer and also controls the triangle and the mouse. To use this device, one holds the mouse and moves it to the place of interest. The control unit provides the *x, y, z, pitch, yaw and roll* data to the host computer through an RS-232 serial port at 19.2Kbps, 8 bits, no parity, one stop. The precise interpretation of the incoming data bits is shown in the technical reference manual [Logitech 91].

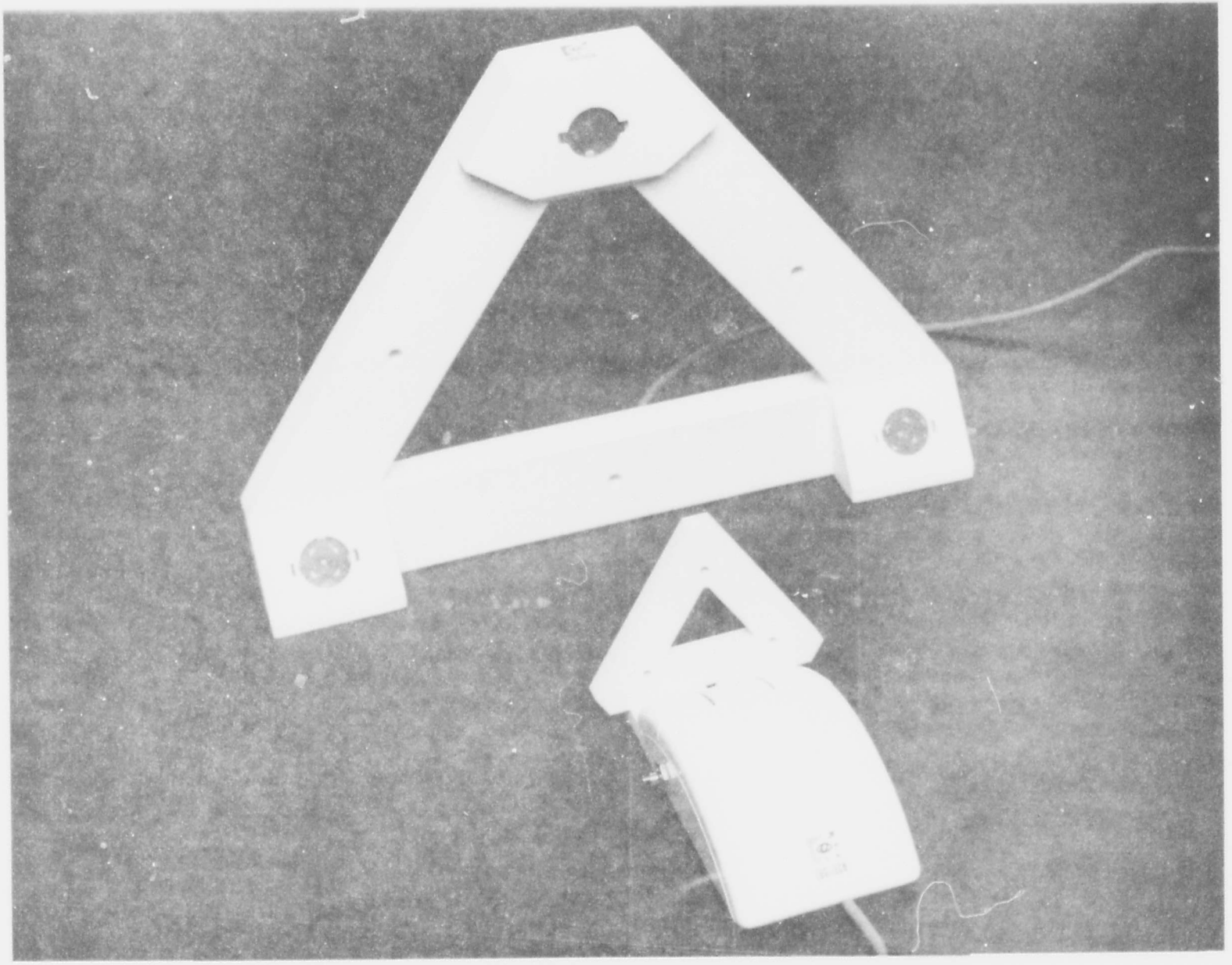

Figure A.1 The Logitech 2D/6D Mouse and Triangle Transmitter

A.1.2 Setup

In normal 6D mode, the triangular transmitter is usually set on a table top. The Mouse is moved in the area in front of the transmitter, since the specified range is a cube 2 feet on each side located directly in front of the transmitter. The active area is determined by the placement of the emitters on the triangle. Each emitter on the triangle emits a 100 degree cone of sound. The top emitter on the triangle emits waves parallel to the floor. The two at the base of the triangle emit waves angled slightly upward. Thus the bottom of the active area is on the same plane as the base of the triangle.

The 6D Mouse can also be used in a head tracking mode. The resolution, accuracy, and update rates are compromised to obtain a larger active area of a cube 7 feet along each axis. The results reported here did not evaluate the head tracking mode for its potential. One possibility for the head tracking mode is to place the triangle on the ceiling facing down. This would minimize the chance that movements of the head would put the receiving triangle out of sight of the emitter.

The mouse is lightweight, and easy to use. Seven-foot cables connect the mouse and the triangle to the control box. Movement is not severely hindered by the tethers due to their length in relation to the small area of operation, although line of sight between all three emitters and receivers must be maintained. If a ceiling mount were to be used with a head tracker, additional length on the tethers might be desirable. If additional mice were to be added, the cables might become tangled and cumbersome.

A.1.3 Manufacturer Specifications

Table A.1: Logitech 2D/6D Mouse Specifications

Model	Logitech 2D/6D Mouse	
Operating mode	*mouse mode*	*head tracker mode*
Degrees of freedom	6	6
Range	2 ft. cube with 8 in. fringe	7 ft. cube
Resolution		
position	0.005 in. in cube, 0.02 in. in fringe	0.02 in. 0.5 degree
orientation	0.1 degree.	
Accuracy	NA*	NA*
Sample rate	NA*	NA*
Update rate	50 Hz max.	25 Hz max.
Latency	19.4 ms	29.4 ms
Tracking speed	30 in. / sec	30 in. / sec
Maximum concurrent sensors	currently: 1 future: 4	currently: 1 future: 4

*NA = not included in manufacturer's specifications

A.1.4 Characteristics and Evaluation

The default coordinate system labels are shown in figure A.2a. However, the coordinates have been transformed in software to the coordinate system shown in figure A.2b. This labeling of coordinates is also consistent with measurements of subsequent VR devices. The Mouse coordinates referred to henceforth will be those of the modified coordinates. Measurements of position are made with respect to the default origin located in the center of the cube of operation. The orientation coordinates will be referred to as *xang, yang and zang*, to eliminate confusion with the pitch, yaw and roll nomenclature. The measurements of orientation are made about a line parallel to each respective axis, the positive angles being counterclockwise rotations when facing the origin.

A.1.4.1 Range

The specified range of operation is a cube two feet on each side, the front face of which is situated 6 in. from the triangle. The bottom face of the cube is on the same plane as the base of the triangle. An additional 8 in. from each face is defined as a "fringe" area of operation which has reduced resolution. This specified range was confirmed with some qualifications.

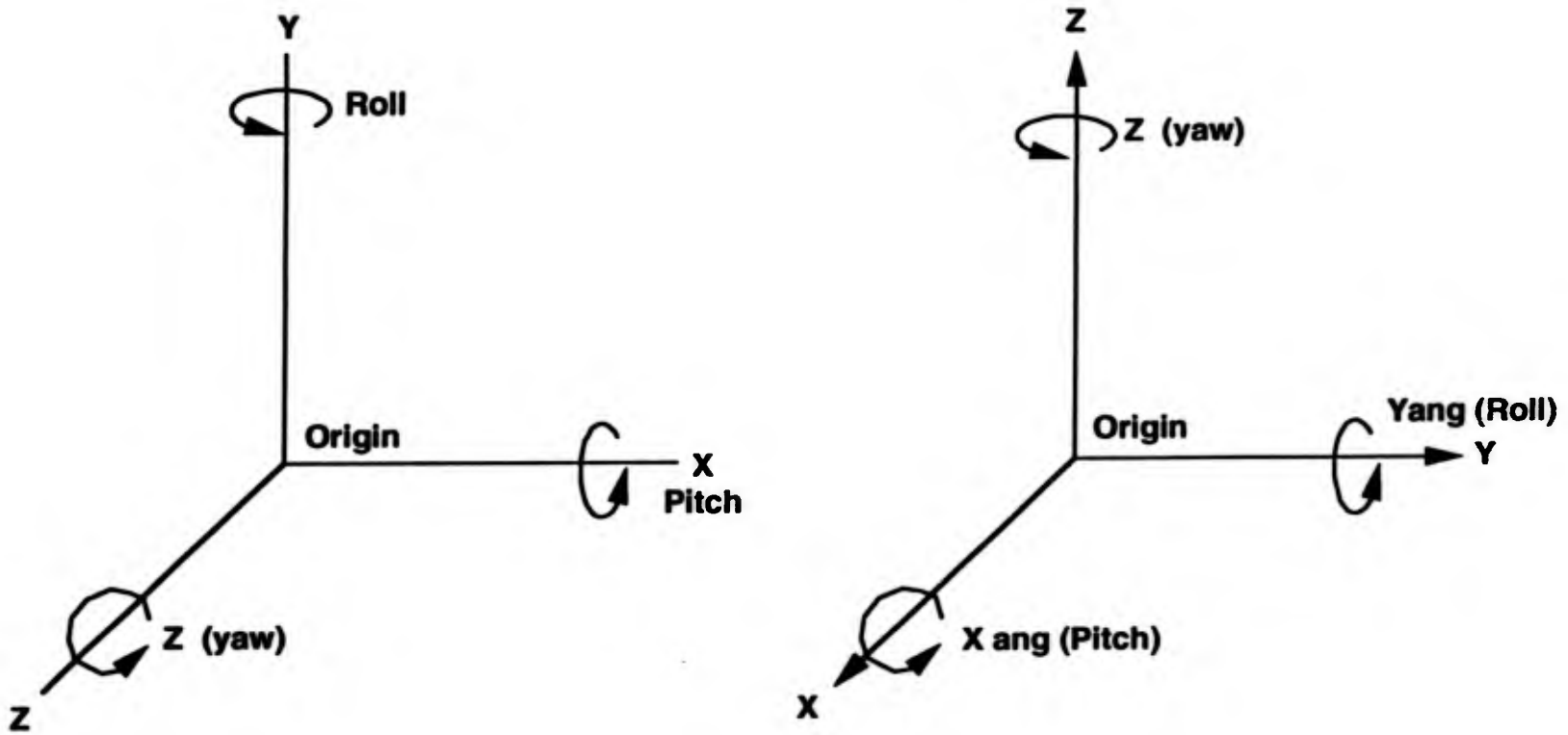

Figure A.2 Logitech Default Coordinate System and Modified Coordinate System

Data was taken over a 3D grid at 12 in. increments. The grid extended 5 ft. from the triangle along the x-axis, -2 ft. to +2 ft. along the y-axis, and -2 in. to +1 ft. along the z-axis. Ten measurements were made at each location, without moving the Mouse in between.

Our determination of the range was based on several criteria. A look at the error vectors in both the x-y and x-z planes from the data set taken above delineated where the measurements were too far off to be accurate. Secondly, the data taken to plot those error vectors was taken 10 times at each location without moving the Mouse. Excessive standard deviations indicated that a particular point was outside the usable range. Finally, the LED indicator lights flickered when a particular point was in the fringe, or out of range completely. It was noticed that the visual representation of a point in this fringe would jump around, even though what seemed like relatively good position measurements could be taken. Hand plots were made of approximately the positions at which this occurred. The combination of these three tests gives a good sense of the range.

Figure A.3 shows maps position error measurements in the four x-y planes in which data was taken. The specified range and the determined range are superimposed for comparison. We confirmed the manufacturer's limits of the range along the y-axis. The upper limit of the range along the z-axis was also confirmed. However, the range seemed to extend about 12 in. further in the x direction than claimed, and the lower limit along the z-axis also seemed to be about 12 in. more than expected.

The range in the head tracking mode was not investigated. Dynamic performance was not considered, and neither was the change in resolution at the fringe.

A.1.4.2 Repeatability

The 2D/6D Mouse was tested for repeatability in the plane of the base of the triangle as well as above it, using the same test bed described above. Two trials of ten measurements were taken at each of six positions. The mouse was moved around in between each measurement and then carefully returned to its original position. Three positions were within the specified range and three were outside of it. The orientation of the mouse was kept such that the plane of the mouse triangle was parallel to the plane of the triangular transmitter. The average and standard deviation for each position were calculated.

The repeatability of the 2D/6D Mouse was within the manufacturer's specifications. The average standard deviation of all three position coordinates was less than 0.04 in. The average standard deviation of all three orientation coordinates was 0.2 degree. Of the other three positions outside the specified range, two were beyond the specified fringe area. The average standard deviation of the three position coordinates was 0.08 in. and the average standard deviation of the three orientation coordinates was 2.6 degrees. A qualitative look at

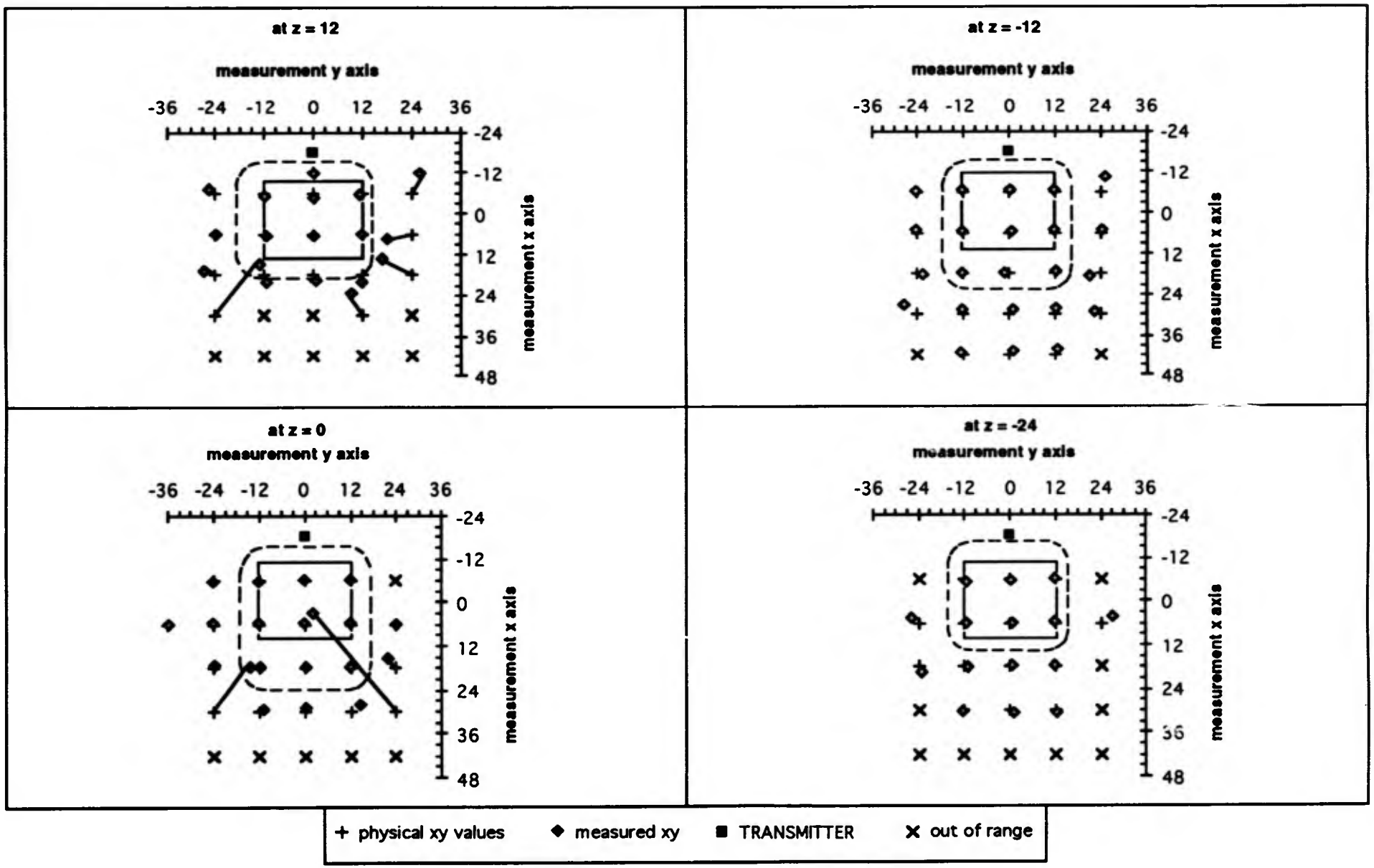

Figure A.3. Map of Logitech Mouse Range of Operations

the results showed that the orientation coordinates are more likely to vary than the position coordinates.

A.1.4.3 Registration

The registration of the Logitech is shown in figure A.3. The average deviation from expected position is 0.8 in. ranging from 0.2 in. to 1.7 in. The average deviations along the x, y and z components respectively are 0.4 in., 0.4 in., and 0.3 in.

Angular registration was only monitored exactly in one plane (z = -12 in.). All orientation values were kept at 0 degrees. The range of variation was -1.5 degrees to +2.6 degrees, although most measurements were less than one degree different from zero.

A.1.4.4 Angle Calibration

The orientation angles were tested to determine the correspondence of the actual angle and the measured angle. No range of angles is specified in the manual, but based on the physical configuration of the 2D/6D Mouse and the fact that the Mouse must maintain a line of sight with the transmitter, it can be seen that the absolute range of angles is *xang* = ± 180 degrees, *yang* = ± 90 degrees and *zang* = ± 90 degrees. The Mouse was moved in ten degree increments through the range of each angle coordinate independently. The measurements were taken in the area about a foot from the transmitter. The physical configuration of the 2D/6D Mouse made it difficult to accurately track the *xang* through the negative angles, and thus it was measured through the range of 0 to +170 degrees. Estimated accuracy in placing the angles is ±1.5 degrees. Additionally, several known *zang*'s were tested against the measurement results, while the *xang* and *yang* were held at 0 degrees. The position was held constant.

Graphs of the measured angle compared to the physical angle are shown in figure A.4. For each case, the other two angles are plotted to help determine large deviations from their expected value of 0 degrees. (Allowances must be made for error in setup.) The *xang* shows excellent agreement within the tested range. Both the *yang* and *zang* show growing disparity near the limits of their respective ranges. The *yang* absolute limits are +70 degrees and -90 degrees, although significant error occurs at +60 degrees and -80 degrees. Additionally, the LED indicator light on the panel of the control unit indicated that the measurement was in the fringe area beyond +60 degrees. The *zang* also showed error at +80 degrees and -70 degrees. The variation in the upper and lower limits has to do with the position of the Mouse with respect to the transmitter. If the Mouse is to the left of the transmitter, for example, the negative limit of the *zang* orientation will be greater than the positive limit. This fact makes it difficult to put limits on the range, but generally ± 60 degrees should be reasonable expectations for the limits of the *yang* and *zang*.

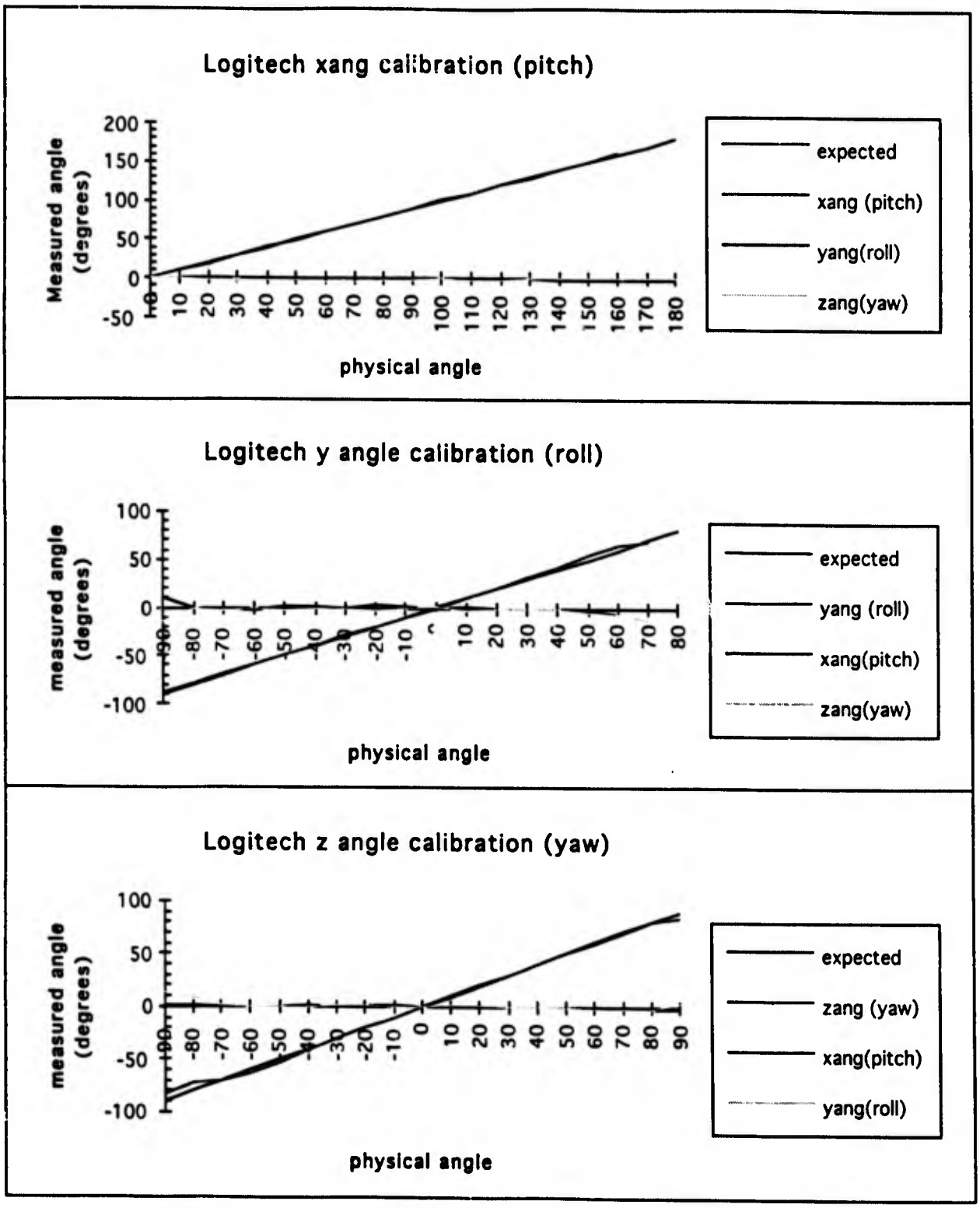

Figure A.4 Logitech Angle Calibration for *xang, yang, and zang*

Figure A.5 shows the plot of the measured *zang* compared to the physical *zang*. It shows good agreement up through ± 56 degrees, with a variation of less than 3 degrees for each one. The *xang* and *yang* varied in this range by as much as 3 degrees also. The measurements at ± 64 degrees and ±90 degrees yielded variations of up to 8 degrees, and variations in the *xang* of up to 2 degrees and *yang* of up to 6.5 degrees. The variation of position for the x and z coordinates was less than 0.1 in. for all angles, and for the y coordinate was less than 0.2 in. except for the angles of ±90 degrees, where the variation was as much as 0.4".

A.1.4.5 Environmental Interference

The nature of the Logitech 2D/6D Mouse demands that the line of sight between the Mouse and the triangle be kept for all three emitters/microphones on each. This obviously precludes all other interference effects in terms of priority. The way in which the Mouse is handled, however, almost ensures that nothing will come in the way of the Mouse and the transmitter. If more Mice are used with the same setup, the possibility of occlusion becomes much greater.

Reflection of the ultrasonic waves off of aluminum plates was tested as a possibility for interference. It was impossible to reflect the waves directly off a plate and still take a measurement, the line of sight to the Mouse still had to be maintained. Several setups were attempted in which a single plate was used in various positions about the Mouse, and none of them had any significant effect on the measurement. Five other setups were made with two reflecting plates, and the measurements were affected only when the plates were parallel. Only the orientations varied in these cases; the position was unaffected. The variation in orientation was on the order of 2 to 6 degrees for each coordinate.

Due to the absorbent nature of most objects in a typical room, and the fact that reflective interference of the ultrasonic waves seems to occur only in resonant situations, it is determined that such interference will not affect the performance of the Logitech 6D Mouse.

The transmitter emits sound at a frequency of 23 kHz. There is a possibility, however unlikely, that other electronic devices may interfere with the Logitech, or vice versa [Meyer 92]. Another possibility of interference with an ultrasonic device is outlined by Meyer, et. al., may result from disturbances in the air [Meyer 92]. This has not been investigated.

A.1.4.6 Jitter

Jitter of the tracked image seems to be a problem when the Mouse enters the fringe area of operation. The image on the display jumps around. Otherwise, the jitter associated with static positioning of the sensor seems to be quite acceptable.

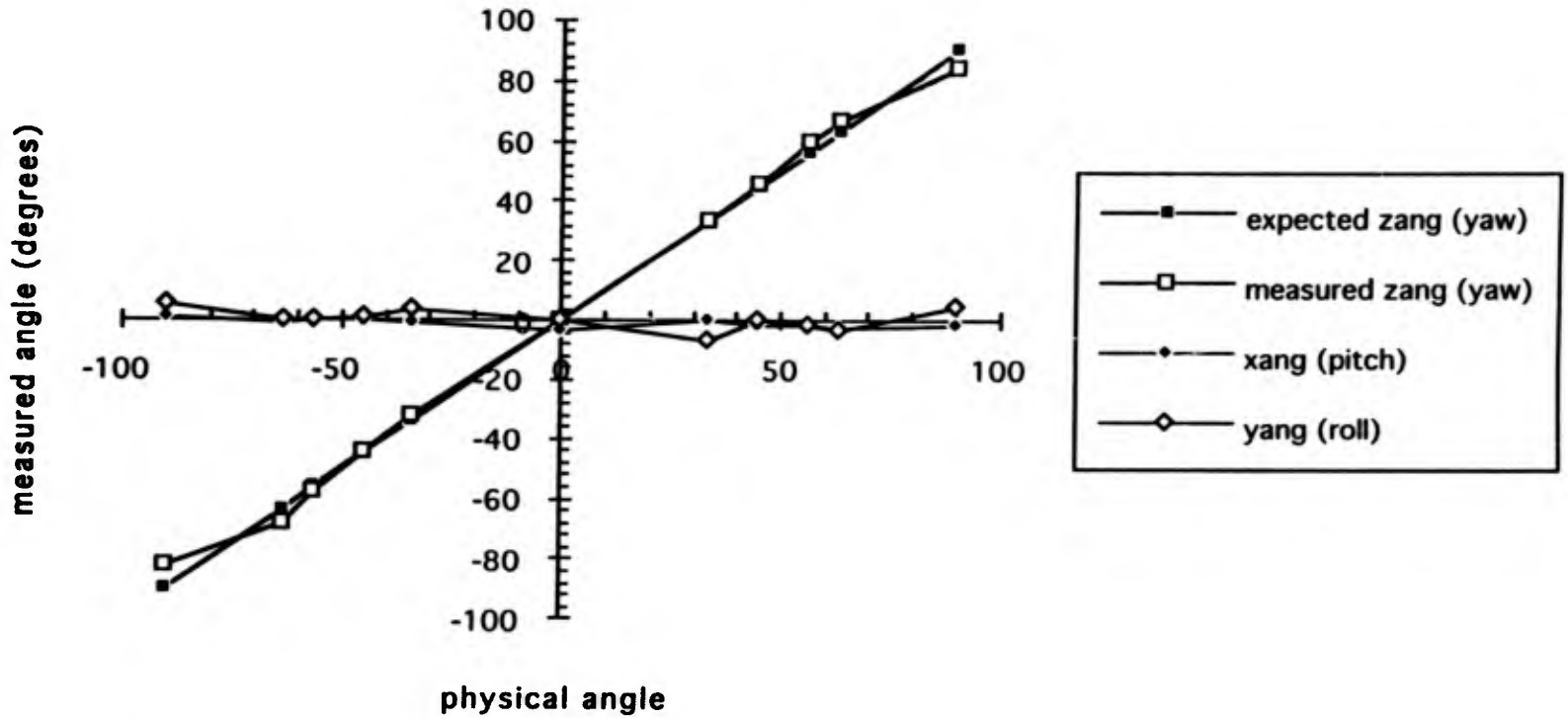

Figure A.5 Calibration about a Fixed Point with Known *zang's*

A.2 ORIGIN INSTRUMENTS DYNASIGHT

The DynaSight Sensor is an infrared optical position sensor manufactured by Origin Instruments, Inc., of Grand Prairie, TX. The device uses an infrared transmitter, the light from which reflects off of passive reflectors (targets) to a receiver which determines only the x, y and z position of the target (figure A.6).

A.2.1 Components

The DynaSight comprises a control unit, a target, and an external power supply. The control unit is an optical transmitter/receiver/signal processor which tracks and calculates the position of the target and returns it in a serial link with the host. The targets come in a variety of sizes and shapes. For pointing, thimbles are provided which are covered with the target material. For head tracking, a target can be placed on the forehead. The target is a circular piece of reflecting material, with sizes ranging from .28 in. to 3.0 in. diameter.

A.2.2 Setup

The transmitter is usually set up on a table, facing the user. The default origin of the coordinate system is a fiducial mark in the middle of the transmitter, and the axes are labeled as in figure A.7. The DynaSight requires line of sight between target and transmitter/ receiver, and it is recommended that the target be at an angle no greater than 30 degrees from the normal to the transmitter. There is also the possibility of using multiple transmitters to track the target in multiple directions, as long as the transmitters do not directly face each other.

No tethers are used with the DynaSight, and the targets are easily mounted. Such a configuration is potentially ideal for mobility and ease of use.

A.2.3 Manufacturer Specifications

The factory specifications are given in the table below [Origin 93-1].

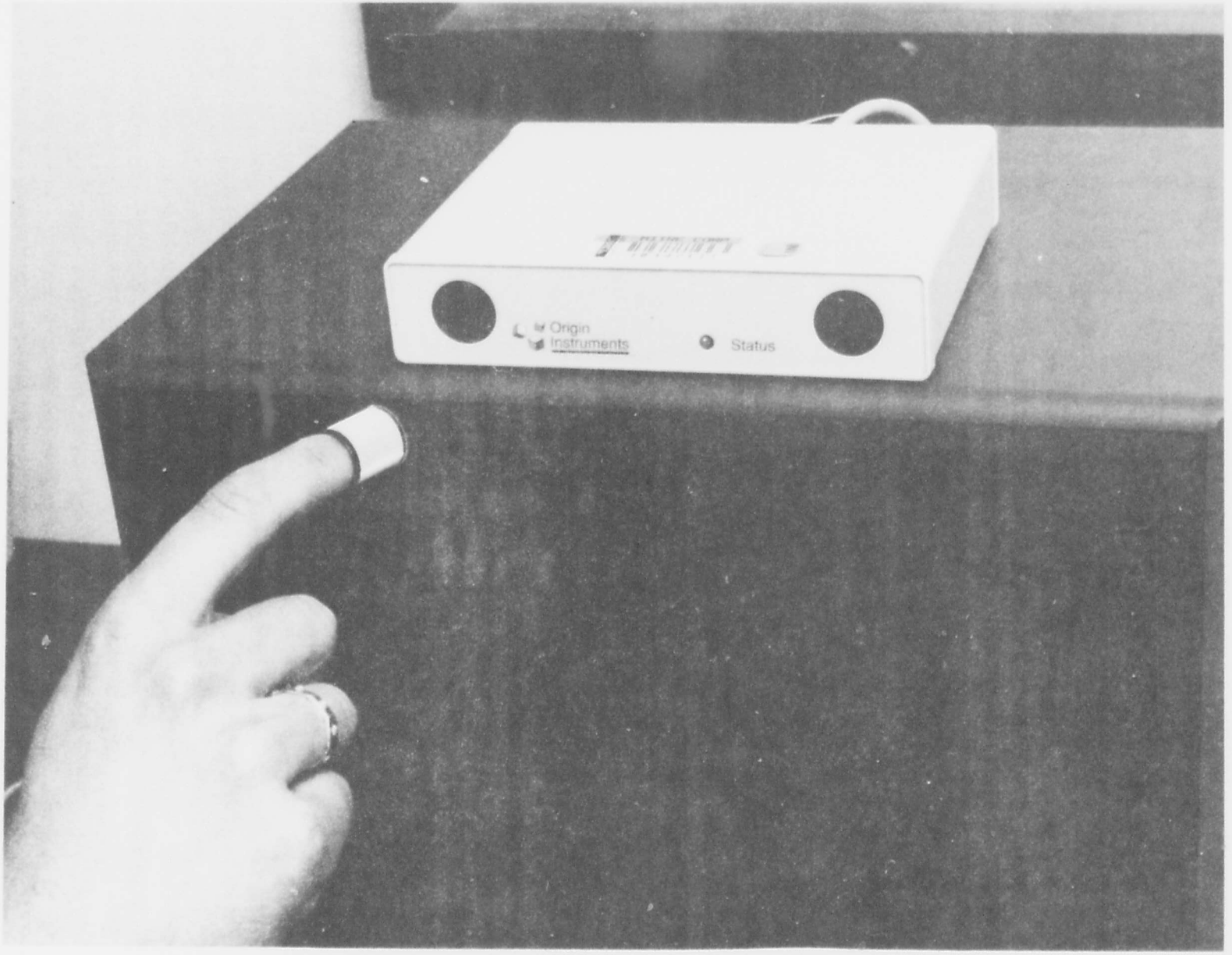

Figure A.6 The DynaSight Sensor and Control Unit

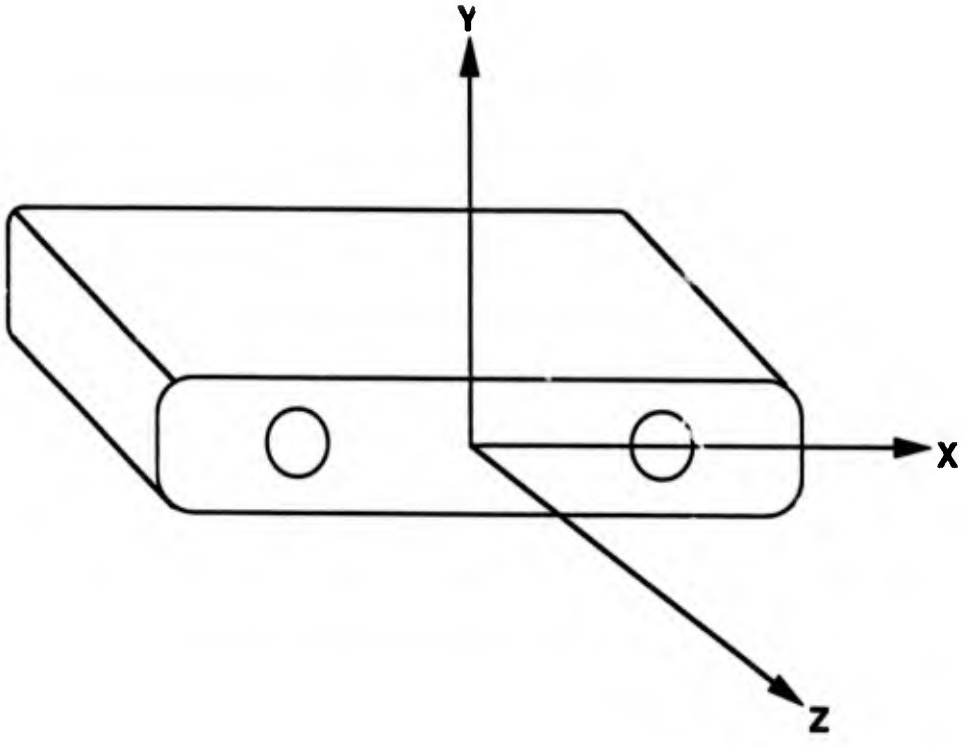

Figure A.7 Coordinate System of the DynaSight

Table A.2. Origin Instruments DynaSight Acoustic Tracker Specifications.

Model	DynaSight
Degrees of freedom	3
Range	.3 ft. to 5.0 ft. within 55 degree cone, with .28 in. target
Resolution	.004 in. cross range .16 in. down range
Accuracy	.04 in. cross range .16 in. down range
Sample rate/Update rate	30 Hz nominal, 33 Hz maximum
Latency	16 - 30 msec
Lock on delay	300 msec
Maximum concurrent sensors	1

Resolution and accuracy are RMS values for the .28 inch target at 31.5 inches from transmitter. These values vary as one gets closer and farther from the transmitter, and are also dependent on the size of the target. The range varies from .3 ft. to 5 ft. for a .28 in. target and 3 ft. to 18 ft. for a 3-in. target. There are several modes of operation which enable a selectable baud rate and thus affect latency.

A.2.4 Characteristics and Evaluation

A.2.4.1 Range

The range of the DynaSight was confirmed to be a cone approximately 55 degrees across. Actual measurements showed that it was nearly 60 degrees in the horizontal and 56 degrees in the vertical, based on the out of range indication of the LED indicator on the front of the transmitter. A 0.75 in. diameter target could be tracked with no problem up to about 4.5 ft. or so from the transmitter. Other target sizes were not tested for linear range.

A.2.4.2 Repeatability

The repeatability was tested as a standard deviation over 10 measurements at each of 6 position. The DynaSight was moved around and returned to its previous position between each measurement. Two trials were taken at each position.

The repeatability of the DynaSight is excellent. The maximum standard deviation of all the position coordinates is 0.09 in. The average standard deviation is 0.03 in.

A.2.4.3 Registration

The registration of the DynaSight is not monitored over its full range. An error vector diagram, figure A.8, shows that each measurement is off by an inch or less at each specified location. It appears that the default axes may be slightly skewed with respect to the transmitter, which may be part of the cause of the misregistration.

A.2.4.4 Environmental Interference

The DynaSight, like the Logitech 6D Mouse, must maintain a line of sight between the transmitter and the target. In addition, the target must be kept within an angle of at 50 degrees from normal to the transmitter.

The DynaSight suffers from reflective interference off of various objects within the room. Normally, the transmitter will track the strongest target within its range. If the target is too close to the transmitter, the system loses track. The receiver in this mode is saturated. When the distance to the transmitter is increased, the ability to track a given target will decrease. Spurious reflections from other objects such as watches, glasses, and even the polarizers over the projector lenses cause the DynaSight to lose tracking momentarily. The effect is compounded by the fact that it takes 0.3 sec to lock on to the target again once tracking is lost.

A.3 ASCENSION FLOCK OF BIRDS

The Flock of Birds, manufactured by Ascension Technology Corporation, of Burlington, VT is an electromagnetic position / orientation sensor. It is capable of tracking up to 30 receivers simultaneously by a single transmitter.

A.3.1 Components

The system comprises one or more Ascension Bird electronic units, (one for each receiver being tracked) connected via a Fast Bird Bus (FBB) and a transmitter. The normal transmitter has a range of ± 3 ft. MITRE has purchased an extended range transmitter which has a range of ± 8 ft. Receivers that are smaller than 1 cu. in. measure the pulsed DC magnetic field and the Bird electronic units calculate the position data. Since each unit has its own computer, the measurements of many birds can be done simultaneously.

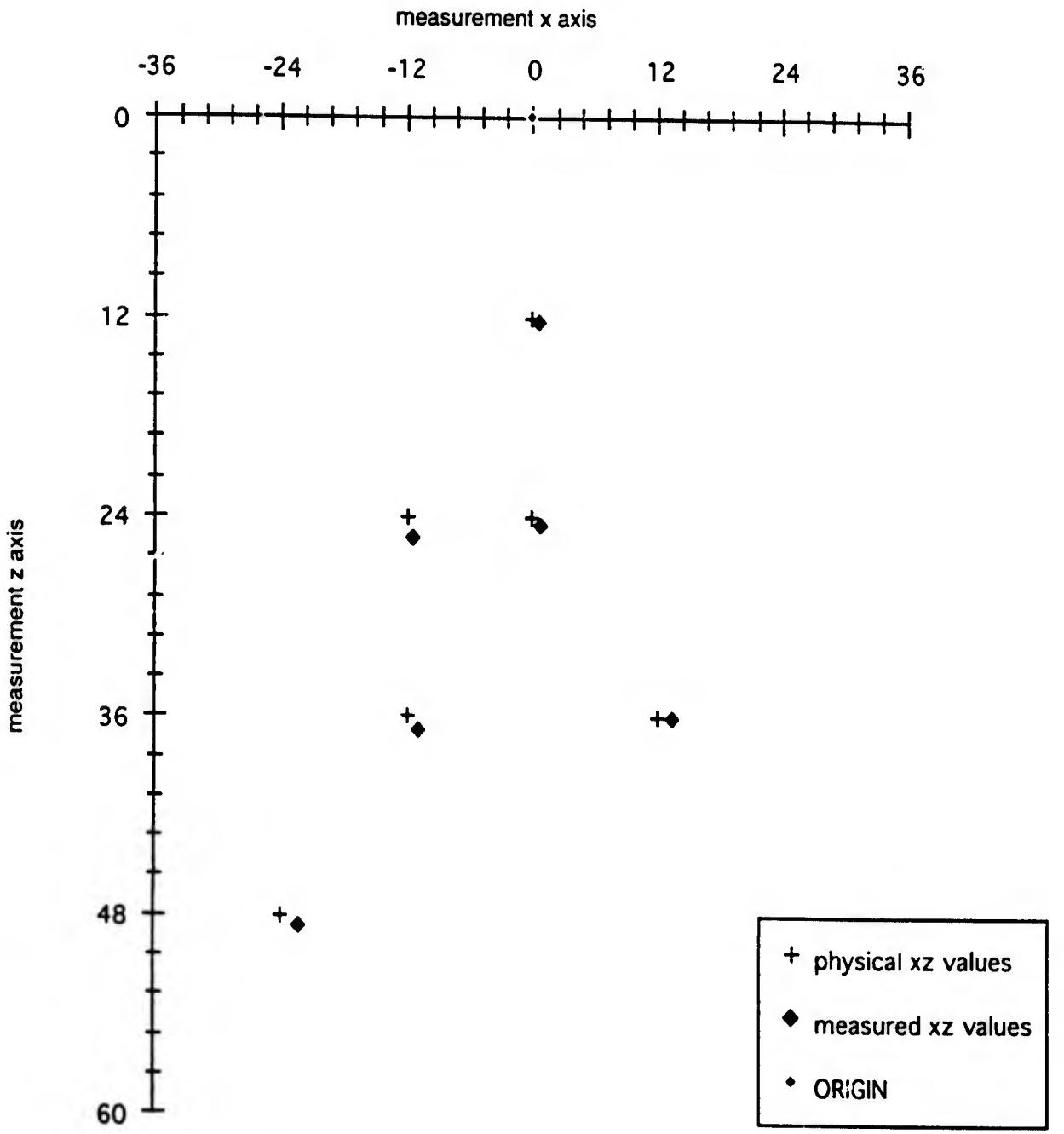

Figure A.8 DynaSight Error Vectors in *xz* Plane

A.3.2 Setup

The Flock of Birds is very versatile. The configuration can be changed to suit the needs of many different applications. User determined parameters include, but are not limited to: number of birds in the flock, selectable baud rate, selectable measurement rate (10 - 144 measurements per second), position data format, operating hemisphere of the transmitter. Changing these parameters changes the performance of the bird. For example, adding birds at the same baud rate considerably slows the responsiveness of the bird receivers. These interdependencies make it difficult to characterize the bird completely in terms of its operating parameters.

The receivers are small and lightweight. They are attached by a 10 ft. cable to the electronics unit. This is long enough for ample movement of the four cables in the same area. However, the cables tended to get tangled very easily.

The external range transmitter is a 12 in. cube. The default origin of the system is in the center of this cube, with a right handed coordinate system, (positive z axis points down, positive x axis points out), as shown in figure A.9. The position angles are positive in the counterclockwise direction when facing the origin along the axis. They will be referred to as *xang, yang, and zang* respectively, to eliminate confusion with the nomenclature. The transmitter was set on the floor, in front of the intended operating area, and both the front and upper hemispheres were used for measurements. The location of the transmitter may be changed in order to minimize the interference effects of the environment. The receivers have been mounted on the DataGlove and other devices with either Velcro or plastic screws.

A.3.3 Manufacturer Specifications

The factory specifications are given in the table below [Ascension 92].

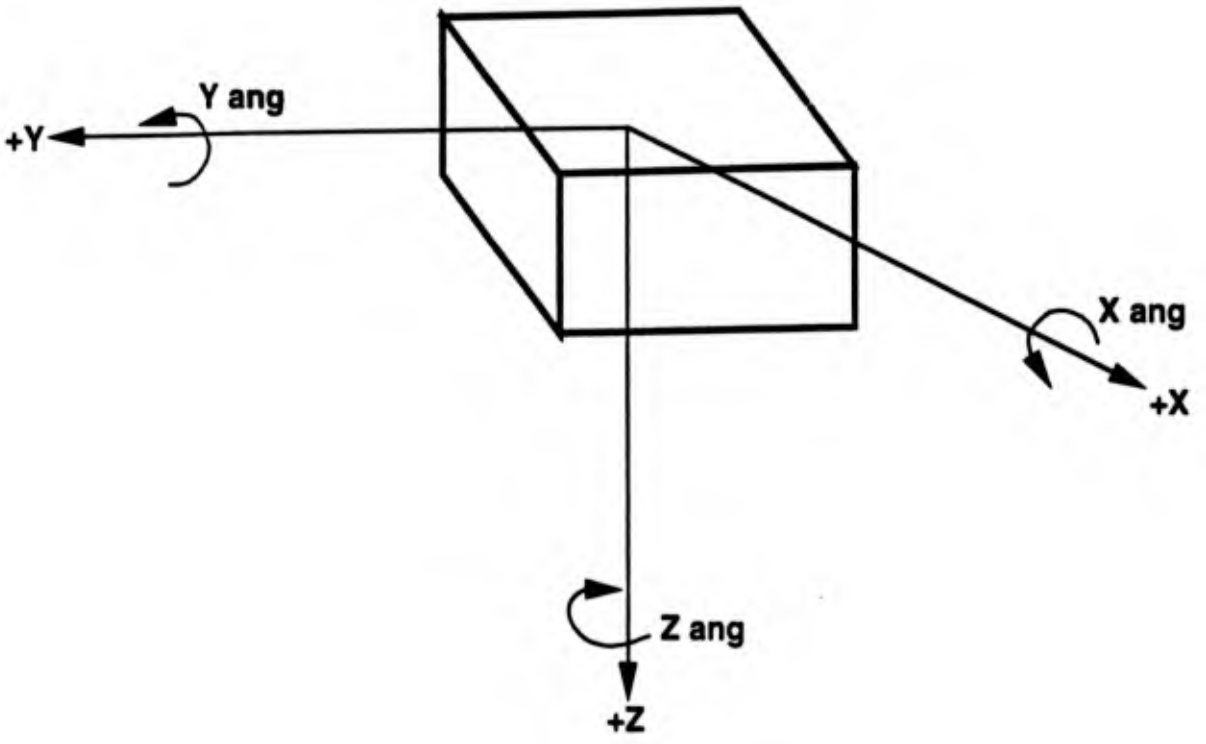

Figure A.9 Diagram of Bird Default Coordinate System

Table A.3. Ascension Technology's Flock of Birds Magnetic Tracker Specifications.

Model	Flock of Birds
Degrees of freedom	6
Range	
position	± 8 ft.
orientation	± 180 degrees, *xang* and *yang*
	± 90 degrees, *yang*
Resolution	
position	0.03 in.
orientation	0.1 degrees RMS
Accuracy	
position	0.1 in. RMS over translational range
orientation	0.5 degrees RMS
Sample rate	100 Hz
Update rate	variable 10 - 144 Hz
Latency	12.25 ms
Maximum concurrent sensors	30

The position and orientation values given are for the normal transmitter with a range of ±3 ft. Since there is no mention otherwise, it is assumed that they are the same or similar for the extended range transmitter. The sample rate for all the birds, regardless of the number in the flock is always 100 measurements/sec. Update rates that are set faster than this will contain duplicate measurements. The latency was not given: However, it is dependent on the baud rate (selectable from range of 2400 to 115K for RS232 interface), the type of record sent, and whether the birds are in point mode (where data is sent only when requested) or stream mode (continuous data sent).

A.3.4 Characteristics and Evaluation

A.3.4.1 Range

The range is difficult to confirm because of all the interference effects within the room. Based on results from various tests below, and taking into account large variations in angle measurements where they should not have occurred, the range of the transmitter appears to be ± 7 ft., given sufficient clearance from objects that cause interference, as explained below.

A.3.4.2 Position Repeatability

The Bird was measured for position repeatability in the general vicinity of the transmitter. Five locations were used. Position one was about three feet directly in front of the transmitter 12 inches off the floor, two and three about five feet to either side of one at the same height, four was about 6 feet from the transmitter 12 inches from the table top, and five was 10 ft. diagonally away from the transmitter 12 inches off the table top.

A marker on the floor indicated the position of the wooden post the bird was mounted on with Velcro. Orientations were not considered, but they should not change significantly except in the zang. Ten position measurements were taken at each location, and the average and standard deviations calculated. The bird was moved around between each measurement.

For positions 1, 2, and 3 the average standard deviation of the all the coordinates was 0.5 in., ranging from less than 0.1 in. to 1 in. The first measurement of some of the sets had to be discarded because of gross error. In subsequent tests, the program was modified to flush the initial reading. Position 4, which was situated directly in front of the steel projector racks in the laboratory, exhibited an average standard deviation of 1.8 in. over its 3 coordinates, ranging from 1.25 in. to 2.6 in.. This is probably due to the interference of the metal at this location. Position 5 was apparently out of range of the transmitter, as it yielded great fluctuations in the measurements and standard deviations of over 40 in.

The repeatability is good in the areas where there would seem to be no "environmental" interference. In measurements for other tests of the range, repeatability looks very good over five measurements taken in the same manner, with standard deviations of less than 0.5 in. in almost all cases in the region of positions 1, 2, and 3.

A.3.4.3 Orientation Repeatability

To test the orientation repeatability, the Bird was set at an arbitrary orientation and moved along the y axis at 3 ft. intervals over 5 positions along the y axis. The x position was approximately 36 in. from face of transmitter and z position approximately 24 in. from floor. Three trials were taken, being as careful as possible not to change the orientation while moving the bird.

The *xang* varied through an average range of 4 degrees, the *yang* varied through 3.2 degrees, the *zang* varied through 6.3 degrees. A graph of the results in figure A.10 shows that the variation is consistent for the *xang* and *yang*. The *zang* is more varied over the three trials because it was more prone to actual variation when the sensor was moved to each testing position. The similarities in the shape of the *xang* and *yang* graphs suggests that the variation may be due to interference. If the angle varied symmetrically along the y-axis, then one would expect that the variation was characteristic of the sensor.

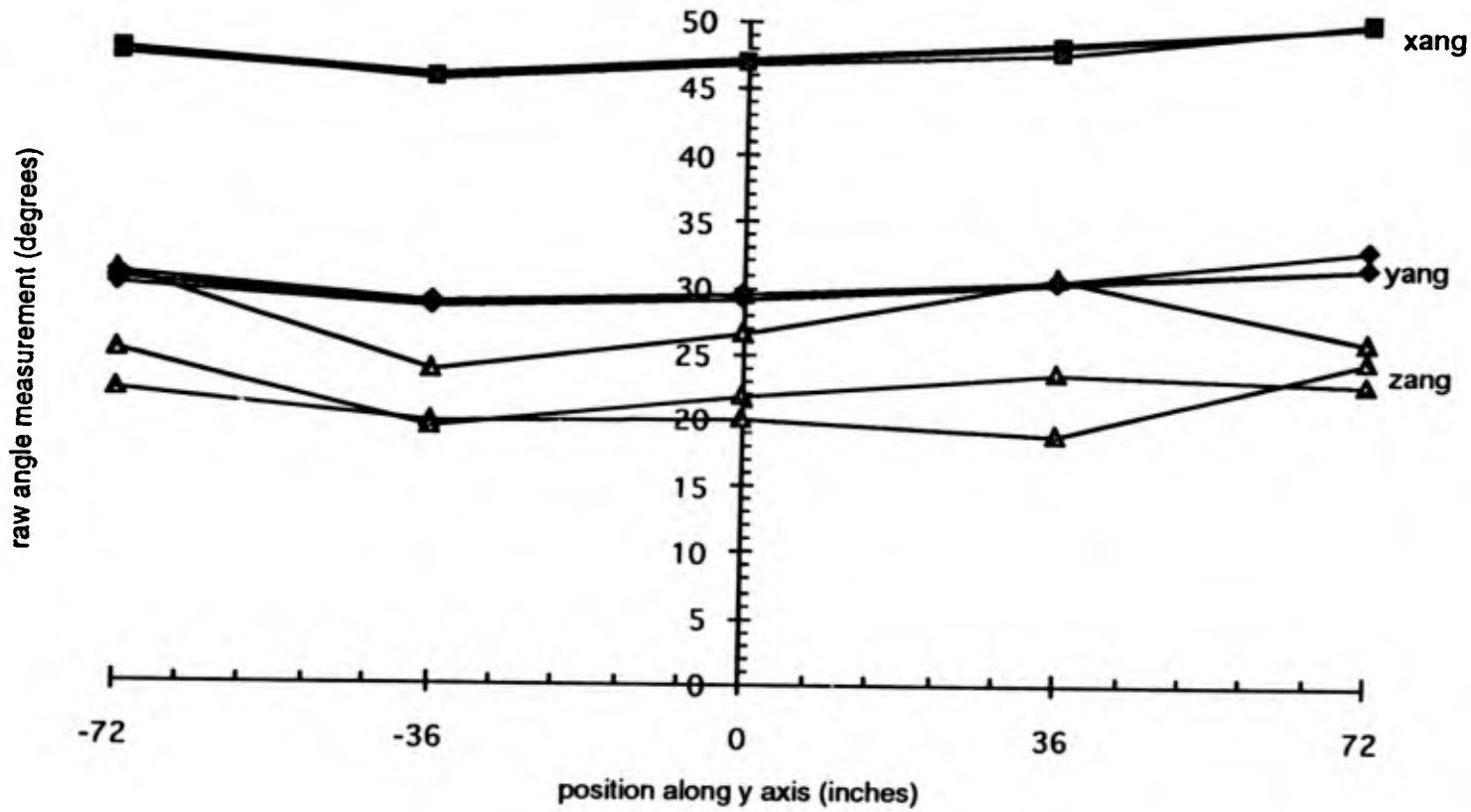

Figure A.10 Bird Test in Constant Orientation

A.3.4.4 Repeatability Over Time

There is a possibility that the measurements vary from day to day, caused by things within the room being moved, added, turned on. This is an area that may need investigation

A.3.4.5 Registration

Bryson suggests [Bryson 92-1] that we can examine the registration by plotting "error vectors," vectors which are the difference between the expected position vector and the measured position vector. In this way, a plot can be generated which demonstrates the way a measuring device varies throughout its range. Figure A.11 shows the error vector plots projected into the x-y plane in two different planes, z = -6.5 in. and z = -32.5 in., respectively (measured with respect to the transmitter origin). The black dots show the physical positions, spaced 36 in. apart, and the white dots show the measured positions. Placement accuracy of the physical positions is estimated to be $\pm$1.5 in. in both x and y directions. The origin is the listed origin of the transmitter. (Placement of the graph axes, y measurements horizontal, and x measurements vertical, is done to simulate the listed transmitter axes as if looking down from the top. The transmitter, of course, is in front of the user.)

Another graph was made of the variance in the z position measurement with respect to the physical z position, versus the x-y distance from the origin. This is shown in figure A.12.

The registration is reasonable directly in front of the transmitter. At the greater height (z = -30.5 in.), the correspondence of the physical and measured values is within a 1 in. to 2 in. of the expected value. At the lower height (z = -6.5 in.), the correspondence is not quite as good. There appears to be a constant x offset of +3 in. which is not apparent in the other plane. There also is the beginning of symmetry about the y - axis, which makes calibration of the sensor easier. (See [Bryson 92-2] for two possibilities relating to the calibration.) In both cases, the registration deteriorates severely at the outer limits of the measurements (>9 ft. from the transmitter). This is outside the specified range, although the manual also states that measurements can be taken up to 144 in. along any axis before the results are erroneous.

The disturbing part about these graphs is that the error vectors are not similar in the two graphs. This may be due in part to distortions arising from the steel reinforced concrete in the floor for the lower plane (z = -6.5), as well as differing positions of other metal objects (tables, chairs). The higher plane exhibits a lesser degree of symmetry, even though the opposite might be expected. The most disturbing part of the results is the changes that occur in the z position measurement. This should be the most stable of all the measurements because it is fixed throughout all the readings. However, the graph shows small deviations of less than 1.5 in. for xy distances up to about 85 in., and then significantly increased error beyond that. This occurs in both z planes.

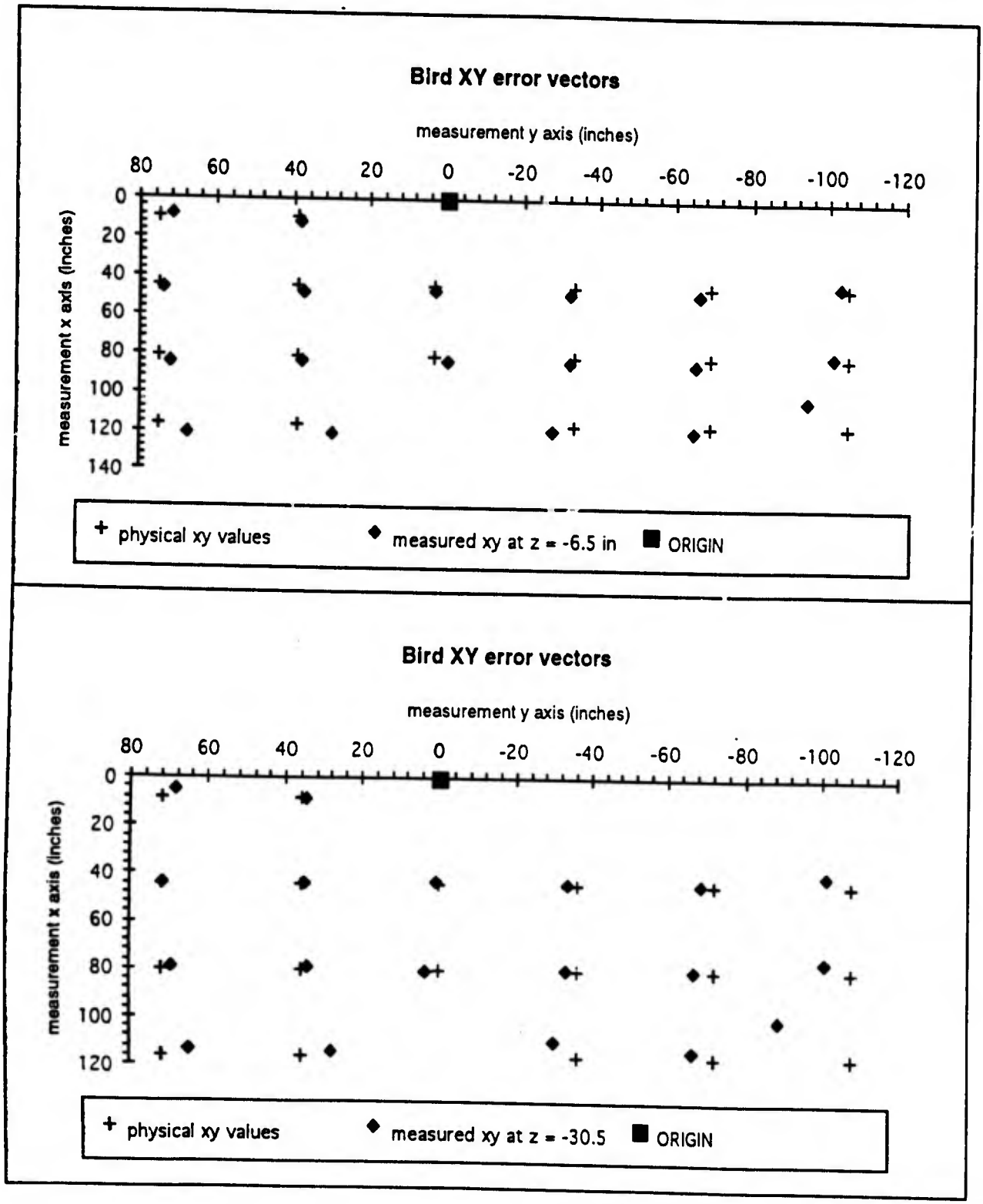

Figure A.11 Bird *x-y* Error Vectors in Two Planes

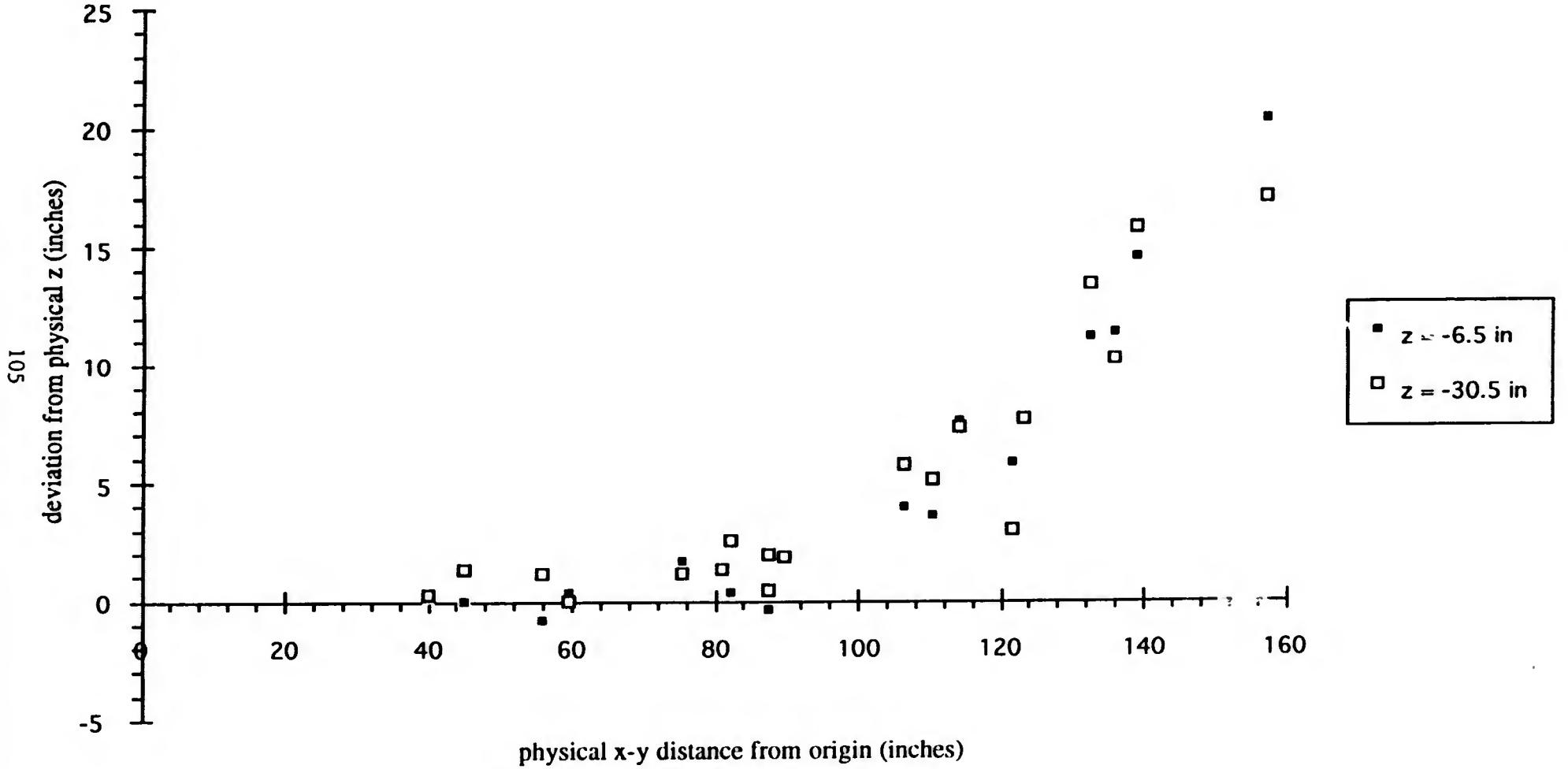

Figure A.16 Bird Variation of z Position Measurement

A.3.4.6 Angle Calibration

The orientation angles were calibrated to determine the correspondence of the actual angle and the measured angle. The position sensor was moved through each range of angles independently in 10 degree increments. The placement accuracy of each angle is estimated to be within 1 degree. Three plots in figure A.13 show the measured angle as compared to the expected angle. Also plotted are the angles which are not being calibrated. They should remain approximately the same, but variations occur due to both physical movement of the receiver and degradation in range.

The correspondence between the physical and measured xang and zang is well within specifications through the entire range of ±180 degrees. Although there are 1 to 2 degree variations from the physical angles, part of this could be attributed to measurement method. The linearity in general is within range and there is minimal variation in the other two angles. The *yang*, shows noticeable deviation from the expected angle at the limits of its ±90 degree range. Additionally, the xang and zang vary significantly at the *yang* = ±60 to 70 degree range. The *yang* measurement is consistently off by several degrees, but this is most likely due to error in the physical measurement. If possible, it might be best to restrict movement in the *yang* direction to within the range of ±75 degrees.

A.3.4.7 Environmental Interference

The Flock of Birds, being an electromagnetic sensor, suffers serious interference due to the presence of metals and other electromagnetic noise in the working environment. The manufacturer's manual recommends that the Birds be operated in an environment as free as possible from both metals and electronic equipment. Given the way in which buildings are constructed, this is nearly impossible.

The interference effects in dealing with the Birds was the greatest single hindrance in their performance. The metallic interference is such that it makes it impossible to know exactly what causes the disruptions and inconsistencies in the measurements, and thus the Birds are difficult to characterize.

A.3.4.8 Metals

There are two problems relating to the presence of both ferrous and non-ferrous metals: induced currents and induced electric fields. We look first at the distortions caused by known objects at different distances, and then can generalize the effect of other objects in the room with similar properties.

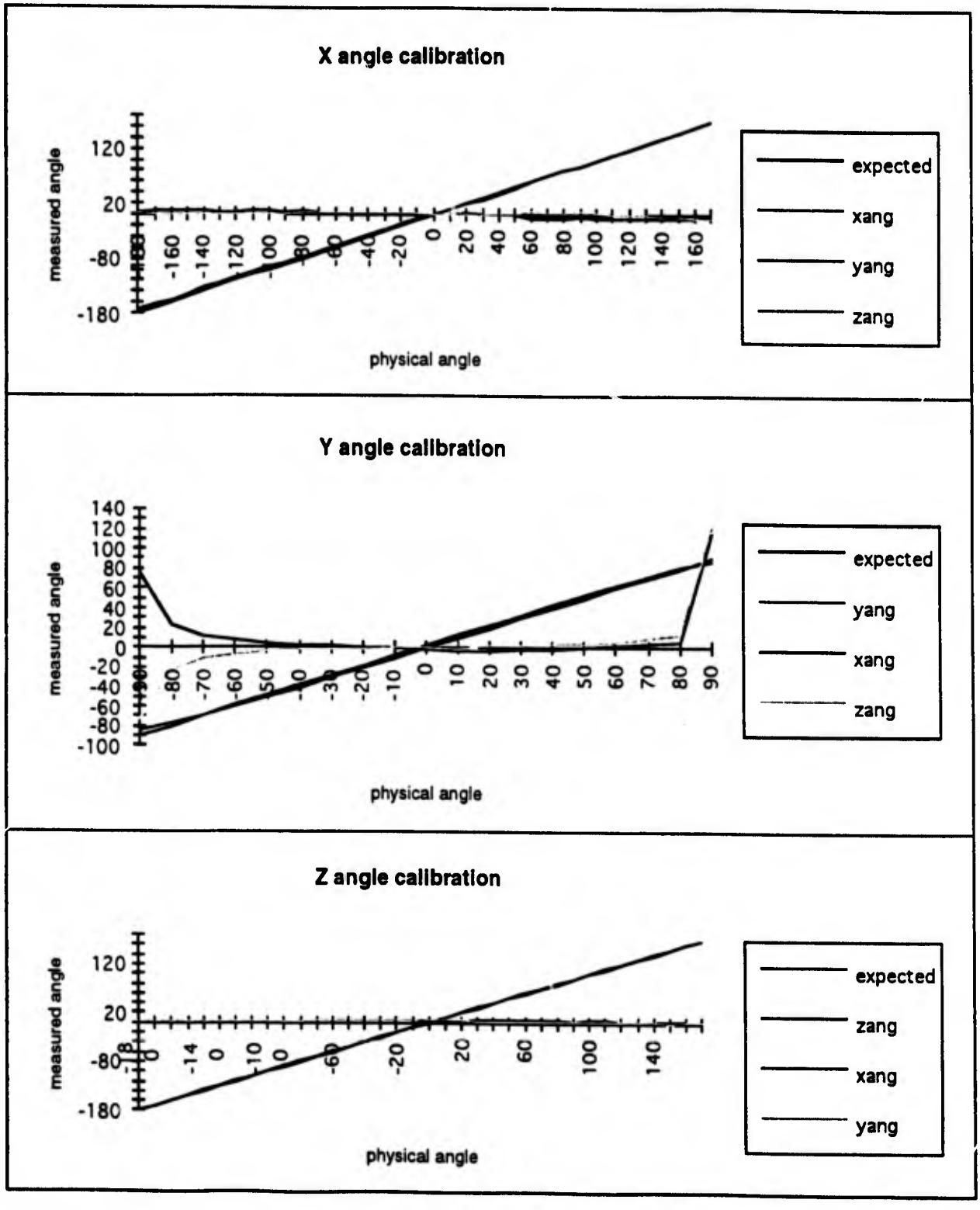

Figure A.13 Bird Angle Calibration for *xang, yang, and zang*

The Bird was set at a fixed position and not moved throughout each series of measurements. To test the interference, interfering objects were placed at distances of 36 in., 24 in., 12 in., 6 in. and 3 in. from the receiver along both the x and y axes. The difference between these interference measurements and a measurement with no interference was then plotted as a function of the distance from the object to the receiver.

Several shapes and sizes of metal were used: a steel plate (2 ft. x 3 ft. x 1/8 in.), an aluminum plate of about the same dimensions, a steel hard drive chassis (1.5 ft. x 2.5 ft.) a coil of wire (45 turns, d = 4.25 in., and a small aluminum plate with a 4 in. hole (9 in. x 9 in.). Several smaller pieces of metal were also used.

Figure A.14 shows the change in magnitude of the position vector exhibited in the presence of the metals. Figure A.15 shows the average change among the three orientation angles in the corresponding situations. Both cases reveal little significant change up to about 12 in. from the sensor. At 6 in. the position change is less than 2 in. and orientation less than 4 degrees. At 3 in. the distortions are quite large.

The effects of the metals are significantly greater along the x axis. The reason is not clear, although it might be related to way in which the transmitter generates the orthogonal fields and fact that the receiver was situated along the y axis when taking the measurements.

The smaller plates were tested in a similar fashion, using various orientations and positions with respect to the receiver. In a quantitative test of the small steel plate (3.5 in. x 4.5 in.), only one significant variation (2 in. variation at 3 in. from the receiver) of the position and no significant orientational variations were revealed. Monitoring the effects of small aluminum plates (2 in. x 3 in., 18 in. x 5 in.) showed that these also created no significant distortion of the measurement, even in cases where the metal was directly on top of the receiver. Furthermore, the effect of the coil of wire explained in the first section was virtually eliminated by introducing a resistor into the coil, which apparently eliminates the induced current.

A.3.4.9 Head-Mounted Display

A head tracker for a head-mounted display (HMD) is one of the key potential uses of the Birds. The n-Vision HMD purchased by MITRE comprises many metal parts as well as high voltage cables to power the CRTs. Thus, one would expect that the use of a Bird on the HMD would create large distortion in the measurements. We investigated the effect of just the metal parts on the HMD, by placing the Bird at several locations: on the top frame, at the back of the visor, and on the visor (see figure A.16). The HMD was not turned on in these measurements. Because the supporting frame of the HMD is fabricated from metal strips, an effect similar to that induced by the coil would be expected. At these three positions, there were deviations of position coordinate of up to 11.1 in., 6.5 in. and 2.1 in., respectively, and

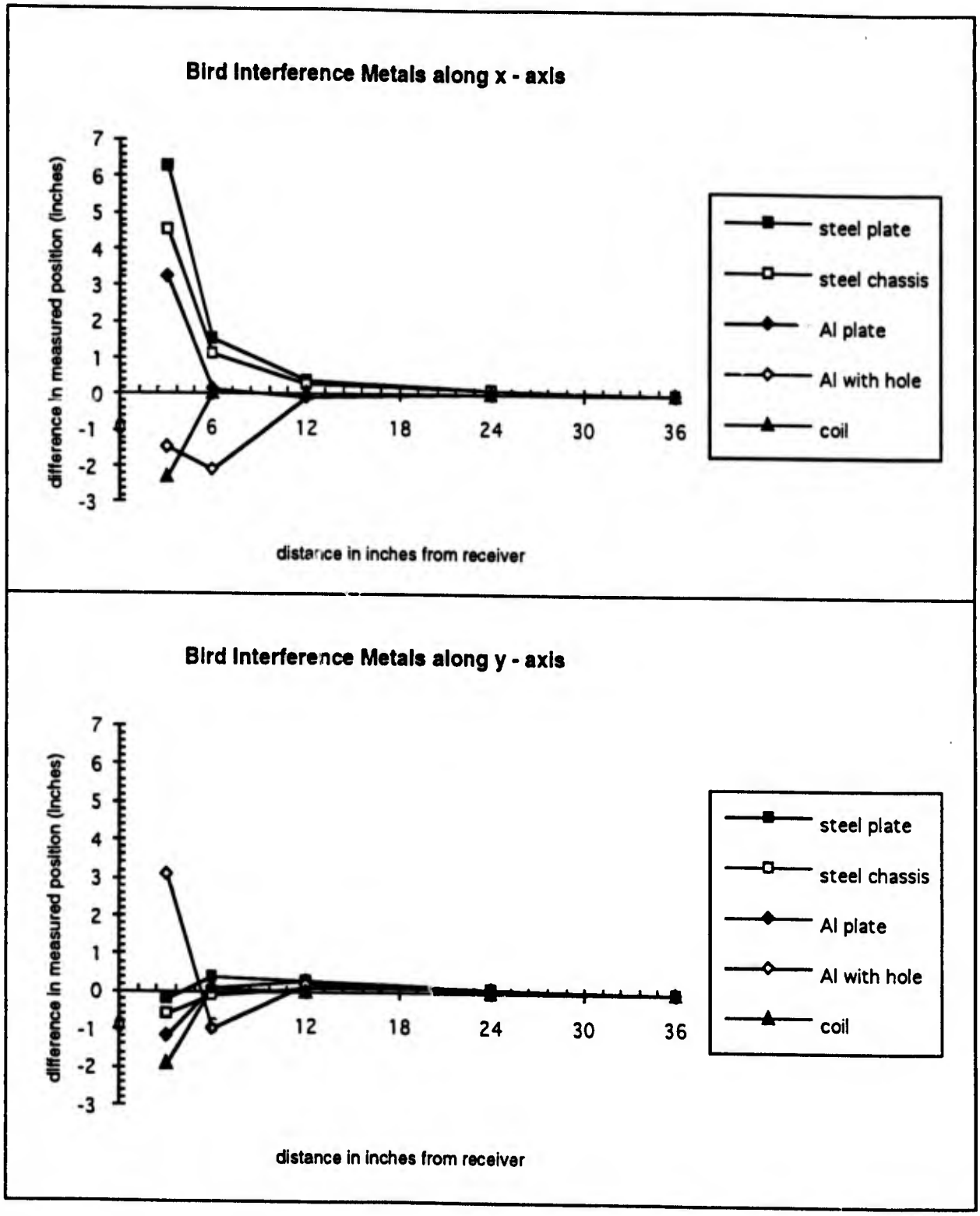

Figure A.14 Interference of Metals on Position Measurement

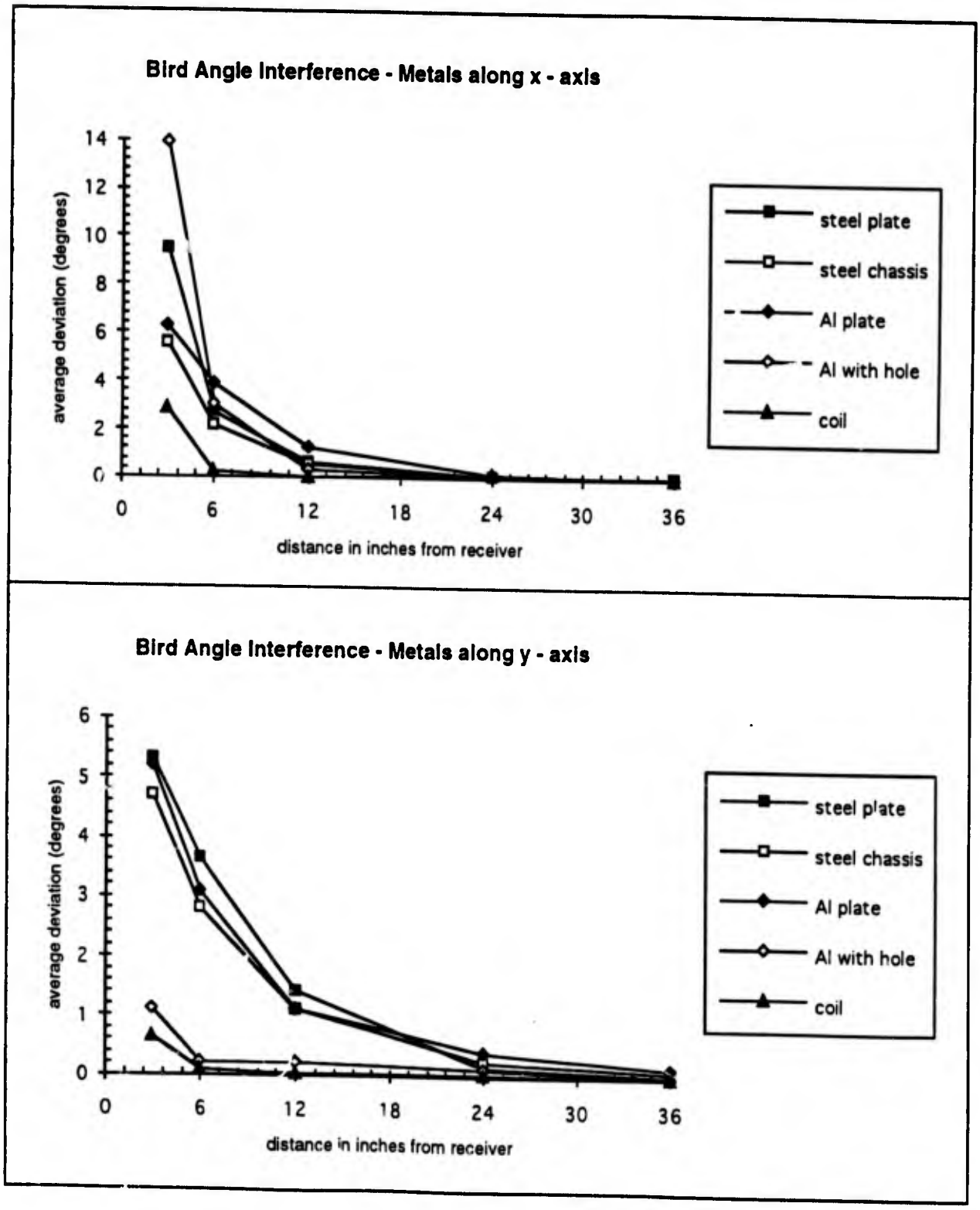

Figure A.15 Interference of Metals on Orientation Measurement

Figure A.16 Photographs of n-Vision HMD

deviations of orientation angles of up to 21.3 degrees, 10.7 degrees, and 3.8 degrees, respectively. Clearly, the position on the visor produces the least distortion. If the HMD cables are sufficiently shielded, or the CRTs pose a constant distortion, the Bird could be calibrated for use on the HMD where relative positions among several receivers is of prime importance. In spite of the seemingly large deviations, the HMD has proved to be operable with the Bird set on the top frame, where the distortion in measurement is greatest. This is because of the adaptability of the human wearing of the HMD.

A.3.4.10 Operating Environment

The setup of the lab is such that there are several factors that will provide significant distortion of the Bird measurements. The projectors for the display are mounted on a large aluminum rack in the center of the room, seen in figure 2.5. The mounting rack acts as a large coil in which a current is induced. The tables all have metal legs, whose effect can be seen in the error vectors of figure A.11, in the third position along the x-axis. The chairs are made of steel frames, which again act as large coils. The floor is steel reinforced concrete, whose probable effect was exhibited earlier. Other input devices, such as the VPL DataGlove, the Exos Dexterous Hand Master, the HMD, as well as computer monitors and cables and electronic component boxes all have electromagnetic fields associated with them. Electromagnetic interaction has been seen with some of them. For example, running the Bird transmitter causes distortion on the monitors. The Polhemus sensor cannot be operated at the same time as the Birds. The Polhemus sensor on the VPL DataGlove, as well as the DataGlove itself, also exhibits some interference when the Birds are running, and vice versa. Additionally, a metal watch band was shown to give distortion to the measurements at a distance of about 3 in.

Obviously an electromagnetic position sensor will not exhibit optimum performance in such an environment. The key then, to minimize the interference effects, is to manipulate the setup of objects in the room and the transmitter so that a working range of operability is created where the transmitter field is dominant over any induced fields and other fields that might be present. This probably entails lifting the transmitter off the floor, and keeping the receiver at least 12 in. from any objects that induce fields. The range will most likely be severely restricted, but at least workable performance will be attained.

A.3.4.11 Jitter

In general, the jitter of the Birds was quite acceptable. There were few cases when the display image seemed to jump around noticeably. Filtering of jitter can be done internally or externally with a net degradation in latency.

A.3.4.12 Number of Birds

The tests above were done with only one Bird operating. In an application with several Birds, several factors come into play. The transmitter outputs a magnetic field whose strength is automatically adjusted to be inversely proportional to the distance of the Birds from it to avoid saturation of the detectors. In the case of several Birds being used, the output of the transmitter is governed by the closest receiver. Thus, any receivers that are much farther from the transmitter will be more prone to error [Ascension 92]. In the measurements above, the Flock was set in a four Bird configuration, and there were always two or three Birds within three or four feet of the transmitter.

The baud rate across the interface cable is selectable. We found that with four birds running, a 9.6 kbps baud rate was slow in tracking all four receivers, and the images jumped across the display, even though this rate was more than adequate for one Bird. The performance of the Birds at higher baud rates is also more prone to error, as there is less time for software filtering of data [Ascension 92].

A.4 POLHEMUS 3SPACE ISOTRACK

The Polhemus 3Space Isotrak, manufactured by Polhemus Corporation of Colchester, VT, is an electromagnetic position / orientation sensor (see figure 2.4). It is capable of tracking up to three receivers simultaneously by multiplexing three transmitters [Polhemus 87]. Since multiple transmitters cannot be on simultaneously, adding additional sensors reduces the maximum sampling rate and therefore increases the latency.

A.4.1 Components

The system as MITRE is using it, is incorporated into the VPL DataGlove Controller. The controller takes the Polhemus position data and combines it with the finger angle data from the DataGloves. If more than one DataGlove is being used, one of the controllers is designated the "master" and it sequences the transmitters so that no two are on simultaneously.

A.4.2 Setup

The normal transmitter has a range of ± 2.5 feet with an extended range to 5 feet with reduced accuracy. Receivers that are smaller than 1 cubic inch measure the magnetic field and the position data is computed.

A.4.3 Manufacturer Specifications

The factory specifications are given in the table below [Polhemus 87].

Table A.4. Polhemus 3Space Isotrack Magnetic Tracker Specification.

Model	Polhemus 3Space Isotrack
Degrees of freedom	6
Range	
position	± 2.5 ft.(5 ft. with reduced accuracy)
orientation	± 180 degrees, *xang* and *yang*
	± 90 degrees, *yang*
Resolution	
position	0.09 in. RMS[to 15 in.]
	degrades linearly to 0.18 in.
	from 15 in. to 30 in.
orientation	0.35 degrees RMS[to 30 in.]
Accuracy	
position	0.13 in. [4 in. to 15 in.]
	degrades linearly to 0.25 in.
	from 15 in. to 30 in.
orientation	0.85 degrees RMS [to 30 in.]
Sample rate	up to 60 Hz
Update rate	up to 60 Hz
Latency	30.7 ms
Maximum concurrent sensors	3

A.4.4 Characteristics and Evaluation

Due to time constraints, the analysis of the Polhemus was much less rigorous and extensive than that of the other position sensors described previously.

The placement accuracy, which is within 0.5 in. can be easily compensated for, by hand-eye coordination. While there was jitter, it could be compensated for by a minimal filtering algorithm on the inputs.

There is a serious problem with metals, both ferrous and non-ferrous in the space of the Polhemus. One sample point x = 14 in., y = -2.2 in., z = -2.5 in. became x = 0.7 in., y = 0.4 in., z = -22.5 in. when an aluminum plate was placed about 6 inches from the glove. This is more than could be reasonably compensated for with hand-eye coordination.

The absolute values measured at any given point in space within a range of 0 to 30 in. are typically within 0.5 in. of the actual value. Standard deviations of static measurements are typically under 1/32 in. However, there were specific ranges, especially between 10 in. and 15 in., where the standard deviation was in excess of 3/16 in. The amount of motion on the screen in this range would be amplified (or diminished) by any scaling factor used to represent the hand. This magnitude of jitter would certainly be noticeable and may be objectionable. This appeared to be common to both the Polhemus and Ascension units.

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.