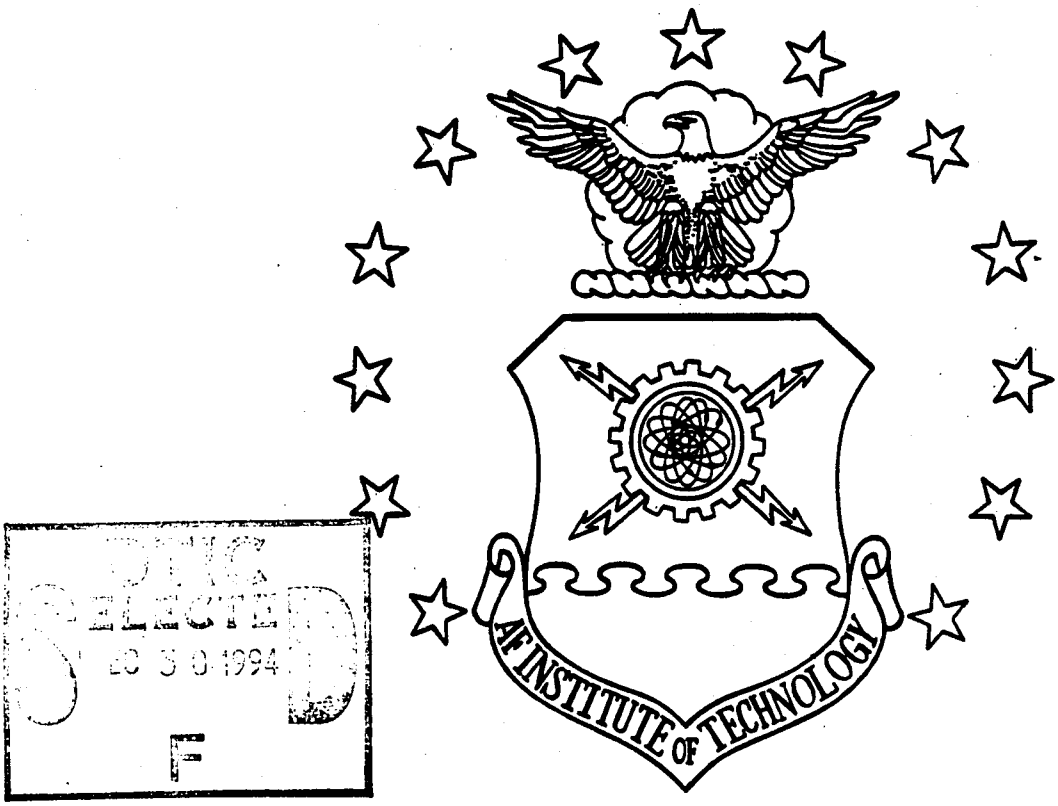


MULTIPLE MODEL ADAPTIVE ESTIMATION AND HEAD MOTION TRACKING IN A VIRTUAL ENVIRONMENT: AN ENGINEERING APPROACH

THESIS

James E. Russell
Captain, USAF

AFIT/GCS/ENG/94D-21

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-21

MULTIPLE MODEL ADAPTIVE ESTIMATION AND HEAD MOTION TRACKING IN A VIRTUAL ENVIRONMENT: AN ENGINEERING APPROACH

THESIS

James E. Russell
Captain, USAF

AFIT/GCS/ENG/94D-21

Approved for public release; distribution is unlimited

AFIT/GCS/ENG/94D-21

MULTIPLE MODEL ADAPTIVE ESTIMATION AND HEAD MOTION TRACKING IN A VIRTUAL ENVIRONMENT: AN ENGINEERING APPROACH

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Science
with emphasis in
Software Engineering

James E. Russell, B.S.
Captain, USAF

December 1994

Acknowledgments

The word *masterpiece* dates back to the Middle Ages. A craftsman (usually called a journeyman) would attach himself to an accepted master craftsman in his field, and then spend sometimes years in intense study, trying to absorb all of the knowledge of his teacher. When at last he was ready, he would create a work that demonstrated his mastery of his craft; this work was called his *masterpiece*. If the master accepted it, then the journeyman became a master.

The process of generating this thesis (and getting my master's degree) bears a strong resemblance to the craftsman's journey to master status. It has certainly been educational, and it has (at times) seemed that it would take years to complete. Also, I have had not one, but many masters under which to study, as well as something not included above – comrades with whom to strive, struggle, commiserate, and, at the last, celebrate. I would like to take a moment to mention some (though certainly not all) of these mentors and comrades. All of you have played parts in bringing this thesis to fruition:

- My thesis advisor and all-around substitute for mom; LtCol Pat Lawlis.
- My domain expert; Dr. Peter Maybeck.
- My gadfly in the staid ointment of engineering; Dr. Robert Eggleston.
- My too-soon-silent committee member, Dr. (LtCol, retired) Phil Amburn.
- My fellow journeymen; John Vandenburg, Bob Caley, Dan Gisselquist, and Jordan Kayloe.

I would also like to gratefully acknowledge the unquestioning love and unending support of my heart, my life; my wife, Elaine. "Who can find a virtuous woman? for her price is far above rubies. ... Favor is deceitful, and beauty is vain: but a woman that feareth the Lord, she shall be praised" [Proverbs31:10,30; KJV].

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3D	Three Dimensional
AFIT	Air Force Institute of Technology
AJPO	Ada Joint Project Office
AMLCD	Active Matrix Liquid Crystal Display
ASCII	American Standard Code for Information Interchange
BASIC	Beginner's All-purpose Symbolic Instructional Code
CA	Constant Acceleration
CGF	Constant gain filter
CPU	Central Processor Unit
CV	Constant Velocity
DOF	Degrees of Freedom
EKF	Extended Kalman filter
FOGMA	First-Order Gauss-Markov acceleration
FOV	Field of View
GNAT	GNU New York University Ada 9X Translator
GNU	GNU's Not UNIX
GUI	Graphical User Interface
ISO	International Standards Organization
HMD	Head Mounted Display
Hz	Hertz (number of operations performed per second)
I/O	Input/Output
LCD	Liquid Crystal Display
MAP	Maximum a posteriori
MByte	MegaByte (1,000,000 Bytes)

MHz	MegaHertz (millions of operations per second)
MIT	Massachusetts Institute of Technology
MMAE	Multiple Model Adaptive Estimator
OO	Object-Oriented
OSI	Open Systems Interconnect
PSD	Power Spectral Density
RAR	Likelihood quotient ($r^T A^{-1} r$)
RMS	Root Mean Square
RR	ME/I likelihood quotient ($r^T r$)
SGI	Silicon Graphics, Incorporated
OMT	Object Modeling Technique [Rumbaugh91]
USAF	United States Air Force
VPN	View Plane Normal
VR	Virtual Reality
VRP	View Reference Point

Abstract

Software engineering tools and techniques were applied to design and implement an application that reduces lag typically present in virtual environment displays. The application was a Multiple Model Adaptive Estimator (MMAE), composed of three Kalman filters, that predicted head orientation one sample period into the future. The environment rendering software used these predictions to generate the environment display. Each of the filters in the MMAE was designed for a different assumed head motion type (benign, moderate, or heavy), which allowed the MMAE to adapt to changes in head movement characteristics.

The use of Ada 9X as an implementation language for a virtual environment applications was also investigated. Ada 9X provides object-oriented features for design and development, and it also offers software engineering support that makes it preferable to C or C++ for the application developed.

Two significant results were produced. The first is a performance baseline for the MMAE that can be used as a benchmark for future research in this area. The other is a performance-based comparison of equivalent Ada 9X and C++ graphics applications in which Ada 9X performance was practically identical to C++. This second result is somewhat surprising, and should be investigated further.

MULTIPLE MODEL ADAPTIVE ESTIMATION AND HEAD MOTION TRACKING IN A VIRTUAL ENVIRONMENT: AN ENGINEERING APPROACH

I Introduction

Virtual environments allow participants – most commonly with the aid of head-mounted display (HMD) equipment, position sensors, and perhaps a data glove – to interact with computer-mediated and maintained environments that can represent, among other things, models or simulations of actual environments. It is believed that this spontaneous form of human-computer interaction will have tremendous benefits. There have already been applications developed to aid in scientific visualization, medical imaging, and training for high-risk tasks such as flying an airplane or working in otherwise hostile environments such as on the ocean floor or the surface of another planet.

There are, however, still several barriers to overcome before virtual environments can fulfill their potential. Current state-of-the-art in graphics technology is insufficient to allow real-time generation and display of photo-realistic images; therefore virtual environments are still in the "cartoon" stage of display realism and lack sufficient auxiliary 3D cues such as shading and shadows. Frame rates are also a problem. A frame rate of 60 Hz (sixty hertz, or sixty screen updates per second) is very good, but can usually only be maintained for the simplest environments; even a modest increase in scene complexity can drive the frame rate to a much less realistic (and

therefore much less usable) 10 Hz. Finally, virtual environment displays tend to lag behind the movements of the participant. This also detracts from the realism and usability of the environment, and has been suggested as a cause for motion sickness observed in virtual environment participants.

The focus of this research has been to apply software engineering tools and techniques to develop an application that reduces the lag typically present in virtual environment displays, and thereby to increase the utility of these environments for training and research. In order to achieve this, a Multiple-Model Adaptive Estimator (MMAE) composed of three Kalman filters was used to predict the orientation of a virtual environment participant's head. This prediction was passed on to the software that generated the environment display, which used it to build the next scene shown to the participant. Each of the filters in the MMAE was designed for a different type of head motion (benign, slow movements; moderate, normal movements; and rapid movements such as in a re-acquisition task) thus allowing the MMAE to adapt to changes in the characteristics of the participant's head movement patterns. The research approach was validated by two studies. The first was a performance study in which subjects taken from AFIT faculty and students were asked to follow the movements of a ball (3D sphere) in a virtual environment that incorporated the MMAE-based approach, as well as a single Kalman filter predictor and no predictor. The other study was a language comparison between two implementations of the software; one in C++ and the other in Ada 9X.

1.1 Problem Motivation

The fundamental idea behind the three-dimensional display is to present the user with a perspective image which changes as he

> *moves. The retinal image of the real objects which we see is, after all, only two-dimensional. Thus if we can place suitable two-dimensional images on the observer's retinas, we can create the illusion that he is seeing a three-dimensional object. Although stereo presentation is important to the three-dimensional illusion, it is less important than the change that takes place in the image when the observer moves his head. The image presented by the three-dimensional display must change in exactly the way that the image of a real object would change for similar motions of the user's head.* [Sutherland68:757]

In his book *Virtual Reality*, Howard Rheingold states that virtual environments have two defining characteristics. The first is *immersion*: the feeling that you (the participant) are actually present in the environment. The other is *navigation*: the ability to move about in and interact with the environment [Rheingold91:112-113].

The combination of these two qualities makes virtual environments a promise-laden research and application field. Rheingold reports that researchers at the University of North Carolina have worked for several years on a virtual environment application that allows participants to explore molecular forces and create new molecules by interacting with models of various atoms [Rheingold91:26-29]. A related piece of ongoing research reported by Chung is attempting to develop a virtual application that allows doctors to see the path a radiotherapy beam will take as it passes through the body, as well as what tissues it will affect. This will allow a doctor to determine a beam position and direction that will maximize effect on cancerous tissue while minimizing risk to healthy tissue [Chung92:193].

Simulation has been and continues to be a major player in virtual environment technology. Simulators have proven to be a cost-effective training medium that allows individuals in high-risk occupations to gain experience without the risk to life or property associated with using actual

equipment. Simulators are used to train, among others: pilots, air traffic controllers, and professional race car drivers [Ellis91:325-326]. Scientific visualization is another growing area for virtual environments. Scientists are using satellite photography data to create surface models of distant planets that can be explored through virtual environment applications [Ellis91:328-329]. Communications and teleoperations have also benefited from the growth of this technology [Ellis91:331-333].

The United States Air Force and the Department of Defense also maintain an active interest in virtual environments. This interest stems mainly from the simulator technology used to train pilots, but extends into other related fields as well. The Ada Joint Project Office (AJPO) sponsors a great deal of research into the use of the Ada language for virtual environments. This thesis is an example of that sponsorship. The Virtual Environments Interface Laboratory (VEIL) at Armstrong Laboratories (Wright-Patterson AFB, OH), is dedicated to the study and development of applications for virtual environments. AFIT also contributes to the growing body of research in this field. Distributed simulation, synthetic battle bridges, satellite modelers, and virtual cockpits all explore the possibilities of this field.

Virtual environments are a step up from current screen-based display techniques, just as current screen-based techniques were a step up from text-based input/output (I/O) displays. The introduction of the *desktop metaphor* in the 1980s allowed a user to interact with his/her computer in a more natural way. Instead of being expected to memorize arcane commands and myriad options, the user could use a graphically-based input device (a *mouse*) to identify and execute desired commands. The screen, which until

this time had been little more than a text-based I/O device, became a more familiar (and less threatening) *graphical user interface* (GUI); in this case a desk top on which symbols (or *icons*) representing documents and other objects or actions could be placed, manipulated, or discarded [Ellis91:321-322].

Now, with virtual environments, the participant can have a virtual desktop with virtual documents, etcetera, that he/she can manipulate with his/her hands. Again, the driving motivation is to make the interaction more natural for the participant. Virtual environments thus represent a blending of the strengths of the computer and the participant. The ability of the computer to manipulate vast amounts of data rapidly and accurately and present it in a graphical format is being coupled with the natural abilities of the participant to analyze, interpret, and manipulate that data. In the words of Arthur, Booth, and Ware:

> *The underlying motivation in virtual reality is to realistically present 3D worlds to a user so that he or she perceives and interacts with them naturally, thus borrowing from built-in human abilities that evolved from our normal dealings with the 3D world that surrounds us every day.* [Arthur93:240]

The key to this collaboration is the spontaneity and perceived realism of the interaction. If humans are to immerse themselves in virtual worlds and use eyes and hands as the means of interaction, then these worlds must provide feedback that is appropriate, expected, and perhaps most importantly, delivered in real-time.

Meeting this feedback requirement is one of the major challenges in designing virtual environments, but it is certainly not the only one. Perhaps the most immediate problem with virtual environments is cost. Although decreasing steadily as technology and acceptance of this medium improves, it

is still prohibitively expensive for individuals or even small businesses to take advantage of virtual environments. One or possibly two (one for each eye image) high-performance workstations are required to maintain them [Arthur93:243].

Another problem is stereoscopic eye strain. HMDs must be precisely aligned to the participant's interocular distance (the separation of the eyes) or the participant will experience severe eye fatigue from extended use. A related problem is that current HMDs are fairly bulky and heavy, and prolonged use can cause neck fatigue [Ellis91:337].

Feedback concerns, however, are the focus of the majority of current research. An example of a feedback issue is auxiliary 3D cues such as shadows and shading of objects [Arthur93:244]. These additional cues add to the immersive quality of the virtual environment display by helping the participant to determine the relative distance between objects, and other display characteristics. However, techniques for providing these cues are currently prohibitively expensive from a computation standpoint [Foley90:866-873]; using them can drive display performance (as measured by presentation rate) below the usability threshold.

Problems encountered in the area of spatial display limitations (resolution and field of view) speak directly to the need for appropriate feedback to the participant. In immersive virtual environments, the images shown to each eye are commonly displayed on Liquid Crystal Displays (LCDs) that use wide-angle optics to give the participant the necessary peripheral cues. The resolution of these LCDs is typically quite coarse, with a relatively small number of pixels available for display, and the images generated by the computer must typically be stretched in order to cover all of the display

surface available [Arthur93:243]. These factors combine to produce images which are unnatural and blocky. Further, current virtual environments have trouble accommodating the participant's ability to focus selectively, or *fixate*, at a given distance. Normal vision allows humans to set their focus at a specific distance in order to view an object of interest. Objects at this distance are seen sharply, while objects nearer or farther away are blurred. Attempts have been made to monitor the focal length of the participant [Ellis91:337] and adjust the environment display accordingly, but widespread use of this technology is still some time off.

Another feedback problem is temporal display artifacts (lag and low frame rate) [Arthur93:244]. Lag refers to the perceived delay between control events to a system and the system's response (either by display or other activity) to those events. Frame rate refers to the ability of the application software to generate and display new scenes for the participant.

Lag is especially important in virtual environments because any lag in the display of the environment directly affects the participant's feeling of immersion [Arthur93, Friedmann92, Liang91]. Shaw states that a virtual environment must be able to display a new image in less than one hundred milliseconds in order to be considered responsive [Shaw93:292]. Lag has also been linked, at least qualitatively, to occurrences of motion sickness in virtual environment participants. In an article for the *Canadian Journal of Physiology and Pharmacology*, Oman reports that most researchers now regard seasickness, car sickness, airsickness, flight simulator sickness, and other forms as different examples of the same syndrome [Oman90:295]. He also reports on Reason's *neural mismatch hypothesis*, which he feels is perhaps the best known of the current theories on motion sickness:

> *Reason argued that the brain probably evaluates incoming sensory signals for consistency by computing the component of sensory signals that is new and unexpected, given knowledge of ongoing movement commands. The brain is postulated to maintain a "neural store"...that are continuously updated based on experience interacting with the physical environment. As a body movement is commanded, the CNS* (Central Nervous System) *is assumed to fetch from the neural store the associated normally anticipated sensory input....Actual sensory input and retrieved sensory memory traces are continuously subtractively compared. The difference is a "sensory conflict" signal. The specific stimulus for motion sickness...was proportional to the number and magnitude of sensory conflict signals.* [Oman90:296]

Arthur, Booth, and Ware state that lag is probably a more important factor than low frame rate where 3D task performance is concerned. Regression modeling of experimental data they obtained showed that lag accounted for more data variance than low frame rate [Arthur93:261]. However, lag and frame rate are not independent; a low or fluctuating frame rate contributes to perceived lag in virtual environment displays [Arthur93:244]. Bryson and Fisher assert that applications with a low frame rate will have poor interactivity *regardless* of how quickly data is taken from external data sources [Bryson90:105].

1.2 Problem Statement

Lag is a problem that limits the usability of immersive virtual environments by inhibiting their ability to provide a realistic experience for participants. As an example, a study by Kozak, Hancock, Arthur, and Chrysler showed no evidence for transfer of training from a virtual environment to a real-world task [Kozak93:777]. The authors, however, felt that this could be attributed to the current state of virtual environment display technology, and that making the environment display and context

more realistic would improve training usefulness [Kozak93:782-783]. One method of making environment displays more realistic is to minimize and/or eliminate the lag associated with image generation and display. In this way, the environment can be made to show the participant a scene appropriate to his/her head position and orientation in approximately real-time, thereby eliminating one of the current barriers to widespread use of this technology.

The characteristics of this problem are twofold, encompassing both hardware and software concerns. On the hardware side, current hardware is unable to provide position and orientation information at a rate that will allow the environment display to be updated in real-time or even near-real-time. On the software side, current software techniques are not advanced enough to perform the calculations necessary to generate a scene of anything more than trivial complexity and still maintain the frame rate necessary to effect immersion for the participant.

1.3 Problem Discussion

> *"The lag problem, for example,...comes across in popular articles as an artifact of today's systems that will probably be solved by some chip one day soon. It isn't that easy."* [Henry Fuchs, quoted in Rheingold91:34]

Traditionally, *lag* (or *latency*) has been generally defined as the amount of time between the application of a control event to a system, and the system's response to that event [Arthur93:241; Liang91:19; Bryson90:98]. In the case of virtual environments, this may be translated to mean the time delay between a participant changing his/her head position and/or orientation (the control event), and the environment displaying a scene appropriate for that change (the response). But this is not the only possible translation.

Bryson and Fisher divide lag into two types: transmission lag and position lag. Transmission lag is a time delay equal to the time to get data from an external source to the application plus the time needed to update and display the scene [Bryson90:105]. Position lag is a spatial lag that is induced by transmission lag and the velocity of the controller [Bryson90:100]. Bryson and Fisher feel that transmission and position lag are the only important lags in an application; if they are minimized, then the effects of other (unnamed) sources of lag in the system will be negligible [Bryson90:100]. Friedman, Starner, and Pentland take a slightly different approach. They identify three major causes of "problems in synchronization of user motion, rendering, and sound" [Friedman92:57] in their application. These are noise in the sensor measurements, length of the processing pipeline, and unexpected interruptions [Friedman92:57]. This research will use elements from both of these sources to define lag.

Lag Definition. *Display lag* is defined as the time delay between a virtual environment participant changing his/her head orientation, and the display of a scene appropriate for that change; this is also the "human perceptible" lag in the display. If a display is done in real-time (such that the participant cannot perceive any lag), then the display lag is zero. Display lag has three major components: transport delay, image update delay, and unexpected delay. Each of these is discussed below.

Transport Delay. Currently, head position and orientation data are measured by a magnetic tracking device such as a Polhemus 3SPACE tracker or an Ascension Bird system. In order to provide data, the tracker has to generate and sense magnetic fields; perform calculations to determine the sensor location; and transmit the results to the rendering software, usually

across a serial communications line [Bryson90:105; Liang91:19]. The time required for this process to complete is the *transport delay*.

Transport delay is largely a hardware-based problem, but solving it may take more than faster hardware. Bryson and Fisher point out that, with current double-buffered approaches to graphics display, "the graphics will always be at least one frame behind the controller" [Bryson90:105]. This lag is independent of any transport delay; it is instead due to the time required to generate and display the frame corresponding to the incoming data.

Image Update Delay. Another cause of display lag is the number of operations required to generate a new display for the participant. Once the data from the controller arrives, the rendering software will typically perform a large number of calculations in order to build a new scene to be displayed and load it into video memory. We will call the time required to perform this operation *image update delay*.

Image update delay is both a hardware- and software-based problem. Scene calculations typically involve a tremendous number of matrix operations to translate, rotate, and scale environment objects. At the University of North Carolina, special processors for these matrix operations (and other computation-intensive steps) have been used to decrease image update lag [Azuma94:section 3].

Software solutions typically fall into one of two approaches. The first approach involves "guaranteeing" a minimum frame rate by managing the complexity of the scene being displayed. Frame rate is monitored during application execution, and if it falls below a predetermined threshold, the scene complexity is reduced to allow faster rendering. One way to reduce complexity is to alter the display format (for example, using wireframe

renderings instead of solid figures). Another is to reduce the amount of data that needs to be processed. Ellis notes that applications that use the Exos hand (an input device similar to the data glove) often turn off a number of the joints [Ellis91:332]. This idea could be extended – an application could selectively omit non-critical objects in a virtual environment display in order to preserve frame rate. Similarly, the application could render objects in the center of the image at full detail, while rendering objects on the periphery of the image with less detail [Ellis91:337].

The other category involves structuring the application software to minimize software overhead. This is especially true for operating system overhead, although this more properly falls under unexpected delays, discussed below. Azuma and Bishop [Azuma94] report that they used this approach:

> *Special care was taken to use fast communication paths and low-overhead operating systems. Interprocessor communication is through shared memory, across Bit3 bus extenders, or through the 640 MByte/sec ring network within Pixel-Planes5. UNIX is avoided except for initial setup and non-time-critical tasks, like reading buttons.* [Azuma94:section 3]

Another area for overhead reduction is in data complexity management. New data structures such as the octree are helping to reduce the storage and time needed to process virtual environments [Foley90:550-555].

Unexpected Delay. The final source of display lag is *unexpected delays* such as network contention or operating system activity. Though usually infrequent, these are also unpredictable, and can have serious negative impacts on virtual environment displays. One of the more common problems encountered with unexpected delays is missed frames.

Most virtual environments seek to display images at a constant rate; thus each image, or frame, is allotted an equal amount of time for generation and display. In double-buffered systems, two areas of memory, or *buffers*, are used so that one frame can be built while another is being displayed. *Overrun* occurs when the calculations involved in generating a frame take longer than the time allotted for that frame [Performer92:7-2]. The Performer software used in the AFIT graphics lab offers the following three alternatives for dealing with this problem:

- Deny it. The application software will simply display frames whenever they are ready, regardless of how long it takes to generate them. Frame rate is not fixed ; there are no constraints placed on how long the system has to generate a frame, nor is there any notion of a minimum time the frame must be displayed. Performer calls this *Free-Running* mode [Peformer92:7-4].

- Display it. The application will accept the overrun and display the frame for a period of time equal to the normal frame rate. Performer calls this *Floating-Phase* mode because the display phase can "float", becoming out of normal phase if overrun occurs [Performer92:7-4].

- Drop it. The application will reject any frame that has an overrun, and instead continue to display the current frame. At the end of the next period, if a new frame is ready, it will be displayed; if two frames have been generated (the one that wasn't ready before, and a new one), the intermediate frame will be ignored and the most current frame displayed. The length of time a particular frame is displayed will always be an integer multiple of the normal frame period. Performer calls this *Phase-Locked* mode [Performer92:7-5]. Please note that this strategy may result in some frames never being displayed; these are referred to as *skipped* or *dropped* frames [Performer92:7-5].

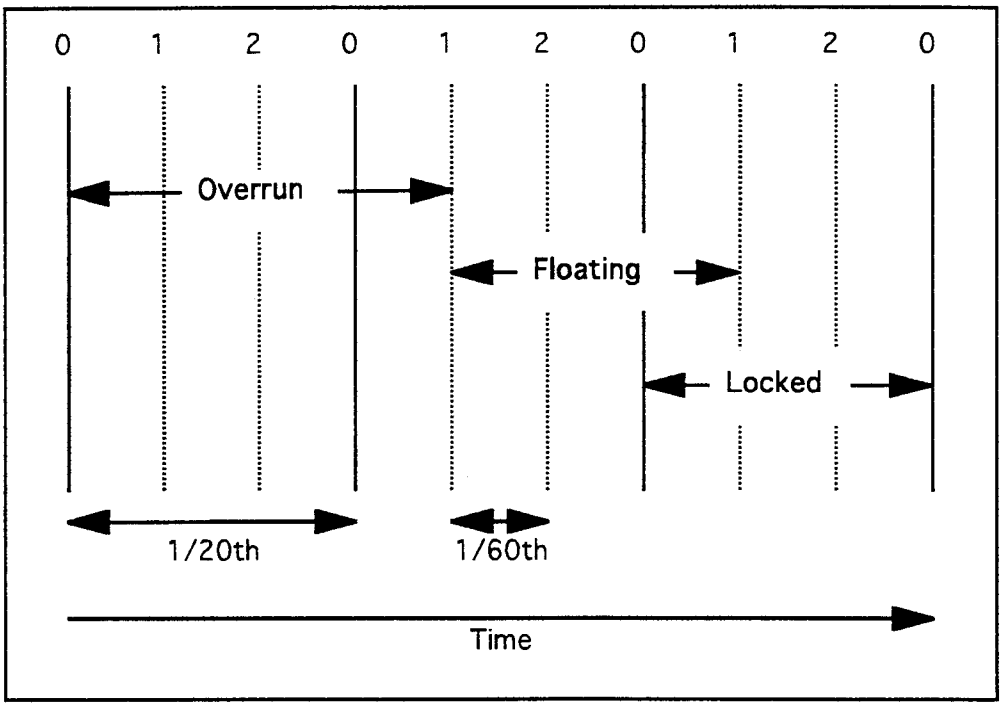


Figure 1: Floating-Phase Versus Phase-Locked Mode
[Performer92:7-3]

The relationship between floating-phase and phase-locked modes is shown in Figure 1. In this figure, the screen refresh rate (shown by the dashed vertical lines labeled 0, 1, 2) is sixty Hertz while the frame rate (shown by the solid vertical lines) is twenty Hertz. Note that, under normal circumstances, a new frame will be displayed every third screen refresh, at the refreshes labeled 0 in the figure.

An overrun is shown on the top left of the figure; generating the frame has taken longer than the 1/20th of a second dictated by the frame rate. The center line in Figure 1 shows what will happen in Floating-Phase mode. The overrun will be accepted, and the new frame will be displayed for the normal amount of time (1/20th of a second in this example). Because this frame is displayed late (at screen refresh 1 instead of screen refresh 0), and because it will be displayed for the normal time, subsequent frames will also be

displayed in the same out-of-phase manner; the phase of the display therefore "floats". Note that if additional overruns occur, it is possible that the display will eventually become in-phase again.

In Phase-Locked mode, frames may **only** be displayed at frame boundaries (the solid lines in the figure). Under this mode, the new frame is not ready at the frame boundary (screen refresh 0; the solid lines in the figure), so the frame currently being displayed is displayed again. When the next frame boundary is reached, the new frame is displayed if it is now ready (which, in our example, it is). Note that it is possible that **two** frames are generated. If this occurs, then the most recent frame is displayed and the intermediate frame is dropped.

1.4 Goals and Objectives

Virtual environments appeal to a broad range of potential users and applications, so a correspondingly broad cross-section of disciplines is represented by researchers. This diversity inevitably leads to a large number of viewpoints about what is important in a given project. The goals and objectives for this research are no exception; however, the main emphasis of this research is on engineering a solution to the real-world problem of visual display lag. With this in mind, the following are the prioritized goals and objectives of this research:

- *Use appropriate software engineering and software architecture skills to design the software necessary to this research.* Specifically, design and implement concrete MMAE and Kalman filter objects for use in predicting head orientation. In line with current software reuse initiatives, the MMAE and Kalman filter should be implemented such that later initiatives may reuse the existing design and software base

developed in this research. This will allow future researchers to leverage their productivity.

- *Examine the appropriateness of enhancing the engineering of the software with an Ada 9X implementation.* By doing parallel C++ and Ada 9X implementations, this research demonstrates the feasibility of using Ada 9X with existing C and/or C++ software bases. It also permits performance comparisons of equivalent applications with and without Ada code. Comparable performance by well-engineered Ada code makes a strong argument for applying software engineering tools and techniques in the virtual environment domain.

- *Compare the performance of the MMAE to Polhemus-only and single Kalman filter applications.* The ultimate test of any engineering process is the product that it produces. Testing and timing studies during implementation, as well as the performance study, will provide data that will characterize the performance of the MMAE by itself and compared to other means of tracking. There are two benchmarks of interest in this area. The first is to determine the benefit of prediction versus no prediction in head motion tracking; the second is to compare adaptive and non-adaptive tracking strategies. The second benchmark is of interest because others [Maybeck94b; Friedman91] have already considered such designs, and the need for adaptivity is seen in performance from these designs. These benchmarks will also serve to validate this application, and provide a performance baseline against which future research efforts may be measured.

1.5 Research Approach

The first step in any research effort is to identify the current state of the art, gather the necessary background information, and scope the problem to be researched. In this case, background information from a large number of areas had to be gathered, and a good deal of time was spent pursuing

reference sources and scoping. Also, an existing software MMAE developed by previous AFIT students [O'Connor92] was studied and enhanced; this provided insight into and experience with the technical aspects of the problem, and aided in software requirements definition. Problems encountered with the prototype, and decisions concerning it, were recorded as Software Problem Reports (SPRs) in order to preserve the experience gained. These SPRs are located in Appendix A of this document.

Once a sufficient knowledge foundation was available, a final architectural and software design for the overall software system was developed. The Rumbaugh Object Modeling Technique (OMT) [Rumbaugh91] was used to identify the static relationships and high-level dynamic behavior of the various components, and in general to encapsulate the decision process. The initial design was then transformed into working code. Most of the graphics software at AFIT uses C or C++ as its implementation language, as does the Performer rendering and display software; C++ was therefore a reasonable language choice for the initial implementation. However, Ada provides excellent support for software engineering goals, and the new version (currently designated Ada 9X) provides the necessary support for object-oriented design and implementation. Also, this application is computationally-intensive, and provides a good comparison of the processing ability of the languages. Therefore, both C++ and Ada 9X versions of the application were developed.

Two studies were then accomplished to validate the research approach. The first involved comparing the C++ and Ada implementations of the software. For this study, performance in terms of maximum frame rate and time required for operations was compared. The other study was a

performance study of the MMAE software. Subjects taken from the students and faculty at AFIT were asked to "follow the bouncing ball" by tracking the movements of a 3D sphere in a virtual environment. Tracking mode was one of the manipulated variables in the study. Data was collected for MMAE prediction, single Kalman filter prediction, and no-prediction (the Polhemus data was used as is) tracking modes. This data was then compared to determine if any performance gain can be realized from using prediction (versus no prediction), or from using adaptive predictors (versus non-adaptive predictors like the Kalman filters).

The final stage of this research was to perform an analysis of the data collected and process used, determine possible directions for future research, and document the research process.

1.6 Method of Analysis

In order to assess the effectiveness of the engineering used in this research, the following analysis methods were used:

Performance Study. A performance study was conducted to assess how well the MMAE predicted head orientation compared to both a simple Kalman filter predictor and no predictor. Subjects were asked to track the motion of a ball in a virtual environment. Each performed several trials; each trial required a different type of head motion to keep the ball centered in the participant's field of view. The results of the test were used to determine the effectiveness of the MMAE in predicting the orientation of the subject's head, and (indirectly) the quality in terms of realism of the virtual display.

Trend Analysis. Data collected during both the implementation and testing phase of development, and during the performance study, were

analyzed to determine how much, if any, improvement was noted with the MMAE predictor developed in this research.

Language Comparison. Finally, the implementations were compared for adherence to software engineering practices and goals. The applications were also measured by performance criteria such as frame rate, time required to do a Polhemus read, and executable size.

1.7 Research Environment

Immersive virtual environments are by nature equipment-intensive, and the environment used in this research is no exception. Fortunately, much of the support infrastructure (in terms of both hardware and software) needed for this research was already in place at AFIT or nearby.

Computer Support. The graphics lab provided Silicon Graphics Incorporated (SGI) workstations with the necessary development environments (C++ and Ada 9X) and graphics support (Performer library [Performer92]) to develop the environment used in this research. Two workstations were used during the research. The first was an SGI 4D with a 40 MHz operating speed running under the IRIX 4 operating system. The other was an SGI Reality Engine^2 running under IRIX 5.2. The ball used in the virtual environment performance study was modeled with Software Systems' MultiGen modeling software [MultiGen94]. SGI's Performer software [Performer92] was used to generate and display the environment.

Magnetic Tracker. Armstrong Laboratory provided a Polhemus 3SPACE tracker [Polhemus90] for use in this research. An illustration of the 3SPACE unit is provided as Figure 2. The Polhemus 3SPACE tracker uses three orthogonal magnetic coils in order to sense the position and orientation

of a source element that can be manipulated by the participant. The sensor accuracy is greatest when the sensor is between four and twenty-eight inches from the source, although separations of up to sixty inches are possible at reduced accuracy. The tracker has a static accuracy of 0.5 degrees Root Mean Square (RMS) and an angular resolution of 0.1 degrees. The maximum output rate of the unit is 60 Hz and can be across either a serial (RS-232) or parallel (RS-488) communications line [Polhemus90:7-2,7-3]. The manual warns that "large metallic objects, such as desks or cabinets, located near the sources or sensors may adversely affect the performance of the system" [Polhemus90:7-3]. During the application performance study, the sensor was attached to the Head-Mounted Display unit described below.

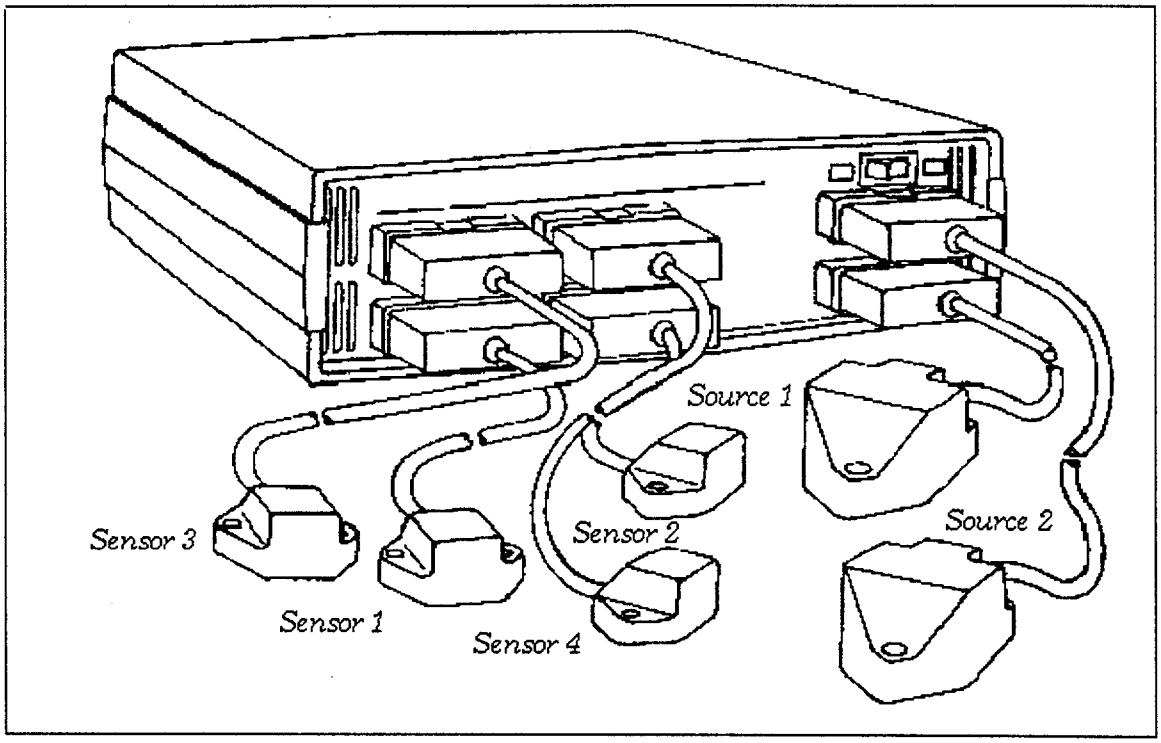


Figure 2: Polhemus 3SPACE Magnetic Tracker
[Polhemus90:1-6]

Head-Mounted Display (HMD). Initial research used a prototype HMD developed at AFIT. While this was acceptable for prototyping work, the weight and general bulkiness of this HMD, combined with its coarse resolution and small field of view, made it unacceptable for the actual performance studies. Instead, a less bulky and more optically pleasing PT-01 Head-Mounted Display [PT01] was used. An illustration of the PT-01 is included as Figure 3.

The PT-01 HMD uses a 420x230 pixel Color Active Matrix Liquid Crystal Display (AMLCD) to show images. The video output format used by the PT-01 is NTSC/RS-170, which is supported by the workstations in the AFIT Graphics lab. The unit allows the participant to adjust the interocular distance of the display between fifty-seven and seventy-two millimeters, and is also eyeglasses-compatible [PT-01]. The PT-01 supports stereoscopic 3D

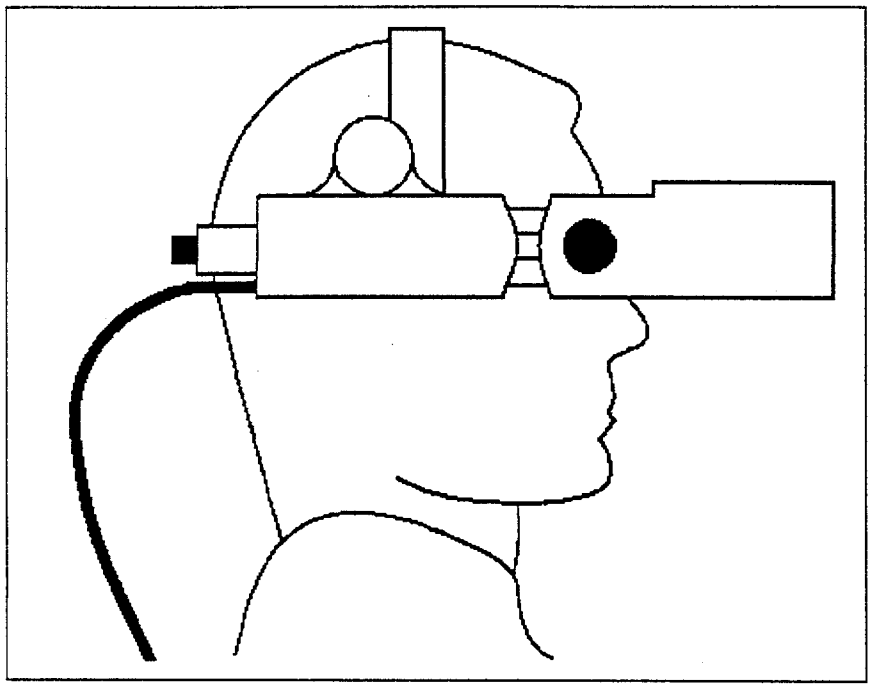


Figure 3: PT-01 Head Mounted Display
(adapted from figure in [PT-01])

imaging and also has audio inputs, but these capabilities were not exploited in this research.

1.8 Document Overview

The rest of this thesis is laid out as follows:

- Chapter 2 is devoted to the large amount of background information gathered during the literature search and thesis scoping process.
- Chapter 3 details the software engineering process used to design and develop the software used in this research, and it provides insight into the rationale used in making various design decisions.
- Chapter 4 discusses the design and implementation of the studies used to validate this research, and presents an analysis of the data collected.
- Finally, Chapter 5 suggests directions for future research in this area.

II Background

The purpose of this chapter is to present historical perspectives and the current state of the art in the various fields, and to identify the specific approaches and techniques used in this research. The reader is encouraged to read completely only those sections that deal with unfamiliar topics; in the others, reading only the subsections on specific approaches used in this research should suffice for background.

2.1 Virtual Environments

Contrary to popular opinion, virtual environments are not a new technology; rather, current applications are the most recent step in the evolution of a technology that can trace its beginnings back several decades. The term *virtual environment* itself, however, is a fairly recent development. Ellis states that it seems preferable to other attempts at naming this technology (i.e., *virtual reality, artificial reality,* and *cyberspace*) because it is "linguistically conservative, relating to well-established terms like virtual image" [Ellis94:17]. He goes on to state that virtual environments can be defined as "interactive virtual image displays enhanced by special processing and by non-visual display modalities, ..., to convince users that they are immersed in a synthetic space." [Ellis94:17]

Historical Development and Discussion. Ivan Sutherland, who is generally accepted as the father of Virtual Reality (VR), built his first head-mounted display in the late 1960s. His landmark 1968 paper, "A Head-Mounted Three Dimensional Display", has proven to be a launching pad for subsequent developments in the field. Sutherland, however, was not

the first man to try to "surround the user with three-dimensional information." [Sutherland68:757] More than ten years previously, an inventor and entrepreneur named Morton Heilig combined sight, sound, touch, and smell into an experience he called Sensorama. His prototype was given a patent in 1962, and then he spent several years attempting unsuccessfully to sell his idea to the entertainment industry [Rheingold91: 49-60]. So it was Sutherland who succeeded in creating a virtual environment, and his success encouraged others, first in the field of computer science, and later in fields as diverse as medicine, psychology, and entertainment, to become involved in the development of this technology.

Of course, not everyone who worked in the field agreed with Sutherland's method of achieving immersion. In the 1970s, Myron Krueger developed METAPLAY, an outgrowth of an earlier experiment called GLOWFLOW. METAPLAY was an interactive environment that used a wall-sized projection screen as its display medium. Participants inside the METAPLAY room were monitored by cameras placed behind the projection screen, and could see their own image on the screen. Pressure sensitive plates placed on the floor were concealed under a polyethylene sheet. Interaction between the participants and the environment was controlled by a facilitator in a nearby control center. Krueger described his creation as an experiment in interaction; he was interested in exploring the effects of such environments on human behavior [Rheingold91:117-120].

Krueger's work was along the same lines as another group exploring virtual environments that didn't require the separation from the real world of Sutherland's HMD. At the Massachusetts Institute of Technology (MIT), a group of pioneers under the direction of Nicholas Negroponte and Richard

Bolt were exploring a concept they dubbed a *Media Room*: a room with wall-sized screens and stand-alone monitors that allowed participants to interact with virtual environments. In the 1970s and early 1980s they developed a demonstration called "Put That There" that allowed a participant to use voice commands and a pointing device to interact with various environment objects displayed on the walls of the room [Rheingold91:95-97].

The 1980s also saw the appearance of another interaction option in virtual environments. Between the two extremes of Sutherland's immersive, equipment-laden environment and the minimal equipment environment of the media room, there emerged another alternative: *augmented reality*. This form of virtual environment used almost the same equipment as an immersive environment; the main exception was a see-through HMD that allowed virtual images to be overlaid on the physical world so that both were seen by the participant concurrently [Azuma94:section 1]. Arthur, Booth, and Ware report that one of the first of these systems was developed by Scott Fisher in 1982. His system allowed a user to view pre-made images stored on videodisk [Arthur93:245]. However, similar systems had been developed much earlier than 1982 for military applications. This type of virtual environment would eventually lead to applications such as Chung's aforementioned radiotherapy beam targeting application [Chung92].

Recent virtual environments research at AFIT has focused primarily on distributed simulation applications that allow more realistic simulations by allowing players from physically separated locations to interact in the simulation. Applications developed in this area include an F-15E cockpit simulator [Diaz94], a satellite modeler [Vandeburgh94], a commander's battle

bridge [Kestermann94; Rohrer94], and an air combat debriefing tool for the Air Force's Red Flag exercise [Fortner94].

Key Concepts. Much of the terminology used in describing various aspects of virtual environments is carried over from the graphics arena. While this is eminently reasonable, it has resulted in some ambiguity of terms. This section will briefly discuss several key concepts used in virtual environments *as they apply to this research*: refresh rate versus frame rate, viewing frustum, and degrees of freedom.

The number of times per second that a picture is redrawn on the display surface is called the *refresh rate* of the display, and is controlled by the display hardware. Arthur points out that a high refresh rate (on the order of sixty Hertz) is necessary to produce persistence of vision [Arthur93:241]. By contrast, *frame rate* is the number of times per second that a scene is updated by the application software. Frame rates as low as ten Hertz are sufficient to produce the illusion of smooth motion [Arthur93:241].

In order to generate a scene, the display software needs to know how much of the environment it should present. This is determined by the *viewing frustum*, an example of which is shown in Figure 4.

The viewing frustum is a truncated pyramid defined by the intersections of the *near* and *far clipping planes* with an infinite *viewing volume* defined by the *horizontal* and *vertical field-of-view* (FOV) and an *aspect ratio* [Performer92:4-6].

Essentially, the viewing volume is an infinitely long pyramid with its apex at a point in space called the *center of projection* [Foley90:230]. The distance between opposing sides of the pyramid, measured in degrees, is the

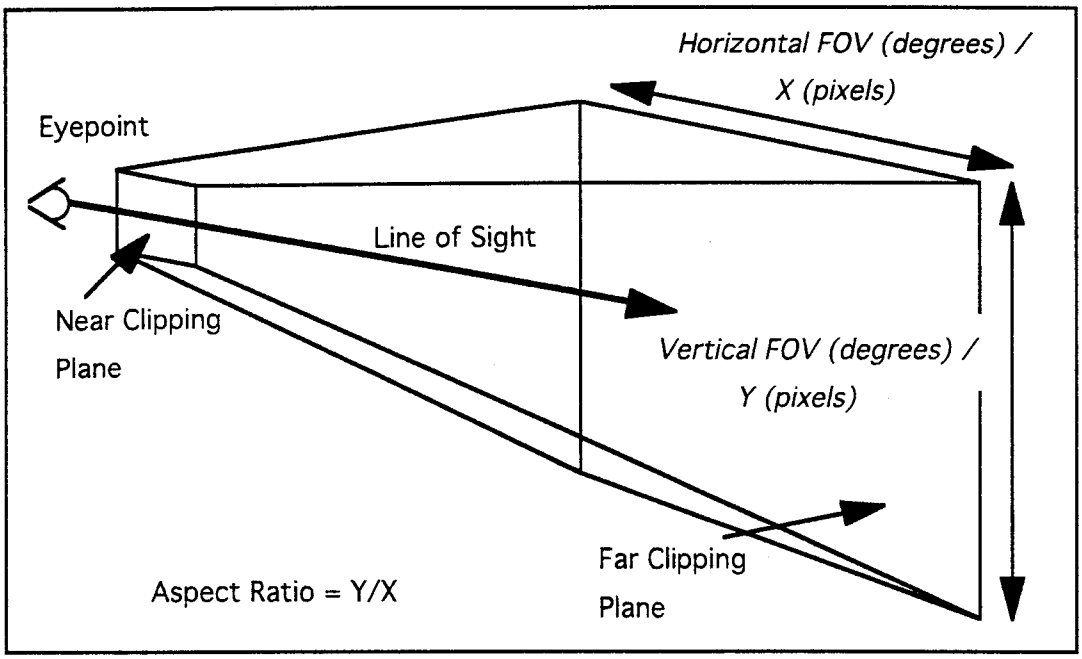


Figure 4: A Typical Viewing Frustum with
Visual Frame of Reference Parameters
[adapted from Performer92:4-6]

field-of-view. Aspect ratio is determined by converting the horizontal and vertical FOV measures to pixels (X and Y, respectively, in Figure 4) and then dividing Y by X. Because the view volume is infinite, we truncate the pyramid with two parallel planes and use the result as our viewing frustum. The plane closer to the viewer's position (the *eyepoint* in Figure 4) is the near clipping plane; the one further away is the far clipping plane [Performer92:4-6,4-7]. When rendering a scene, any object or portion of object that is within the viewing frustum is rendered; anything outside the viewing frustum is ignored, or *clipped*.

The final concept is *degrees of freedom* (DOF), which denotes the ability of the participant to alter visual aspects of the display he/she is seeing. Zero DOF indicates that the viewer is stationary, and can look in only a single direction. If the viewer can move only in a single, straight line (say forward

or backward) but not change the direction of view, then he/she has a single DOF. Most current virtual environments offer the participant six DOF; the ability to alter view position (three DOF) as well as view orientation (another three DOF) [Azuma94; Liang91].

Thesis Approach. The virtual environment used in this research was an immersive environment generated by an SGI workstation, and viewed through a PT-01 HMD [PT01]. The participant's location within the environment (viewpoint) was fixed; however, a Polhemus 3SPACE magnetic tracker attached to the HMD allowed the participant to change view direction freely (three DOF) by measuring head orientation (head position was not considered in this research). The environment itself was comprised of the following:

- Two planes; a green (ground) plane below, and a blue (sky) plane above. The ground plane had a cross-hatch texture applied to it to provide motion cues for the participant. The blue plane had no features, and was provided by Performer through the pfESky command. The viewing frustum was defined such that the planes appeared to intersect and form a horizon line. The overall effect was of hovering slightly above the ground and looking toward the horizon.

- A small crosshair symbol that was placed at the center of the display window. This symbol was either white or black depending on whether or not ball motion was enabled.

- Two balls (one red and the other blue) created with the MultiGen modeling software [MultiGen94]. The balls were the only non-stationary elements of the environment.

- A light source placed directly beneath the participant's location in the scene. The light source was used to give the ball 3D highlights when it moved.

An illustration of the environment is shown in Figure 5.

The environment used in this research is designed to be a minimal environment that will provide the elements necessary to perform the studies. This is in line with the goal of providing a performance baseline for future research. By using this minimal environment, the effects of using more complex (and therefore more computationally expensive) environments, such as a flight simulator, can be more accurately analyzed.

The environment was rendered through the Performer software. This software was chosen for three reasons. First, it is the most commonly used rendering software for graphics work at AFIT, and so was a reasonable choice for compatibility reasons. Second, Performer offers the ability to get real-time statistics on its performance in terms of frame rate and CPU utilization, and these were needed for performance analysis. Finally, Ada bindings exist that allow Performer to be used from Ada applications, thus allowing the development of an Ada implementation of the application.

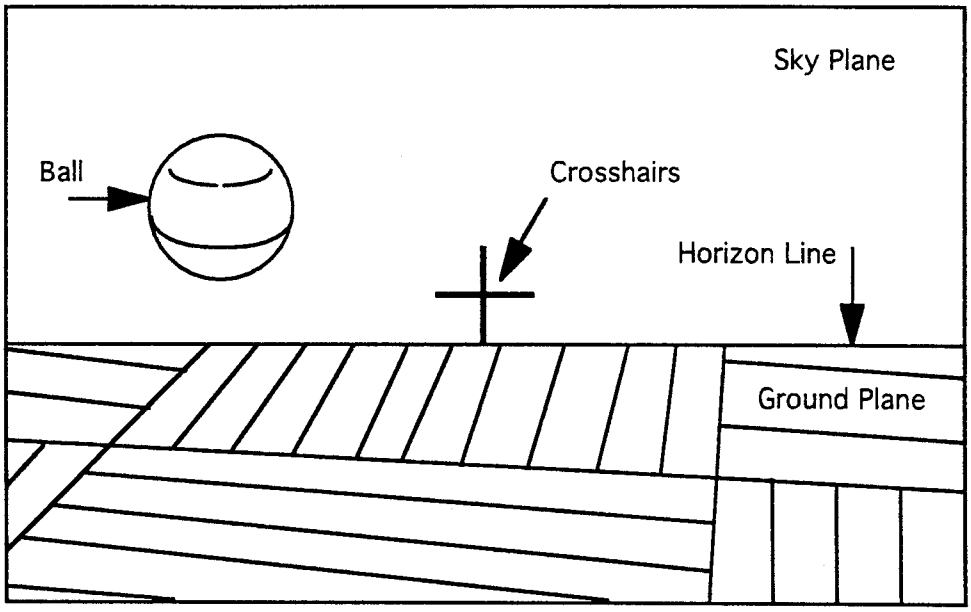


Figure 5: Thesis Virtual Environment

2.2 Kalman Filters

Kalman filters provide a means to estimate state variables that are not otherwise available in a system. Kalman filters can also, under the right assumptions, be used to predict the future state of a system. These properties make the Kalman filter desirable for tracking, monitoring, and control systems, and much research has been done on Kalman filter applications both in the field and at AFIT [Maybeck94b, Friedman92, Chang84].

The discussion that follows presents a high-level overview of the predictor/corrector model and Kalman filters, focusing on intuitive and physical representations. It is not a rigorous mathematical development, although formulae will be introduced as appropriate. Most of the material for this discussion is taken from [Maybeck94a] and the first chapter of [Maybeck79].

The Predictor/Corrector Model. Perhaps the best way to introduce this topic is with an example. Let us assume that you (a somewhat inexperienced sailor) are on a cruise. You wake up in the middle of the night and discover that your navigation equipment is no longer working. You immediately take a reading with a sextant to estimate your distance from shore (for sake of simplicity, we will restrict ourselves to a one-dimensional distance function, as, in the eastward direction). You estimate that you are 100 miles from shore.

You know that the sextant readings will not be completely accurate; errors in the sextant mechanics, atmospheric disturbances, and lack of experience will all affect its accuracy. Further, the readings are just as likely to be too far east as too far west. Therefore, we can characterize the

probability distribution of your reading as gaussian, centered about a mean, μ, of the reading (100 miles), and with a standard deviation, σ, that is dependent on your ability and environmental disturbances. Since you are an inexperienced sailor, you reason that the possibility of your estimate being wrong is fairly large, and that σ is therefore somewhat large on this basis alone.

Using this reading as a basis, you sail for some known time at a "constant" velocity u (plus an error term w, modeled as a white gaussian noise, which accounts for ocean currents or other perturbations,) and then take another reading. As you sail, the probability distribution for your estimate moves according to your motion model (constant velocity u); but it also "spreads out" (the variance increases) because as you travel you become less certain of your exact position (due to the error term w). Just before you take a new measurement, the probability density will have a mean and variance given by the following formulae:

$$\mu_p = \mu + u * \Delta t$$

$$\sigma_p^2 = \sigma^2 + \sigma_w^2 * \Delta t$$

where the subscript p denotes propagated or predicted; σ_w^2 is the strength of the white Gaussian noise w (the height of the power spectral density curve – this describes the power per unit frequency of the noise); μ and σ are the mean and variance, respectively, of your original reading, and Δt is the amount of time that has elapsed. Intuitively, the new mean value is the initial mean value plus the velocity at which you travel multiplied by the amount of time that has elapsed. Similarly, the variance is the initial variance plus the strength of the error term multiplied by the amount of time

that has elapsed. Also, since this prediction is based on a gaussian distribution, it will also be a gaussian distribution.

When you take the new reading, you can use its probability distribution and the probability distribution of the prediction to generate a combined distribution. This combined distribution will have a mean and variance given by:

$$\mu_c = \left[\frac{\sigma^2}{\sigma_p^2 + \sigma^2}\right] * z_p + \left[\frac{\sigma_p^2}{\sigma_p^2 + \sigma^2}\right] * z$$

$$\frac{1}{\sigma_c^2} = \frac{1}{\sigma_p^2} + \frac{1}{\sigma^2}$$

where the subscript c denotes combined or corrected; the subscript p denotes a value for the prediction; and no subscript denotes a value for the new reading.

This is the predictor/corrector model. At each time t_i, you generate a prediction of your state at time t_{i+1} using your current prediction model; then at time t_{i+1} you take an actual measurement and use it to make corrections to your prediction model. At time t_0, you have the choice of either using an actual measurement to start the cycle (as was the case in this example), or using an assumed initial distribution.

Kalman Filter Basics. With the predictor/corrector model as background, we begin this discussion with a system that is driven by known control inputs and monitored by measuring devices that provide information about the system state by measuring the values of some linear combinations of system variables that are corrupted by measurement noise. The state information produced by these measuring devices cannot be exact; there are qualitative and quantitative errors in the system that corrupt the accuracy of

the measurements. Qualitative errors occur because both the system and the measuring devices themselves have error sources. Quantitative errors occur because in all but the most trivial systems the measuring devices cannot access *all* the system variables, and so cannot produce *complete* state information. This situation is shown in the left half of Figure 6.

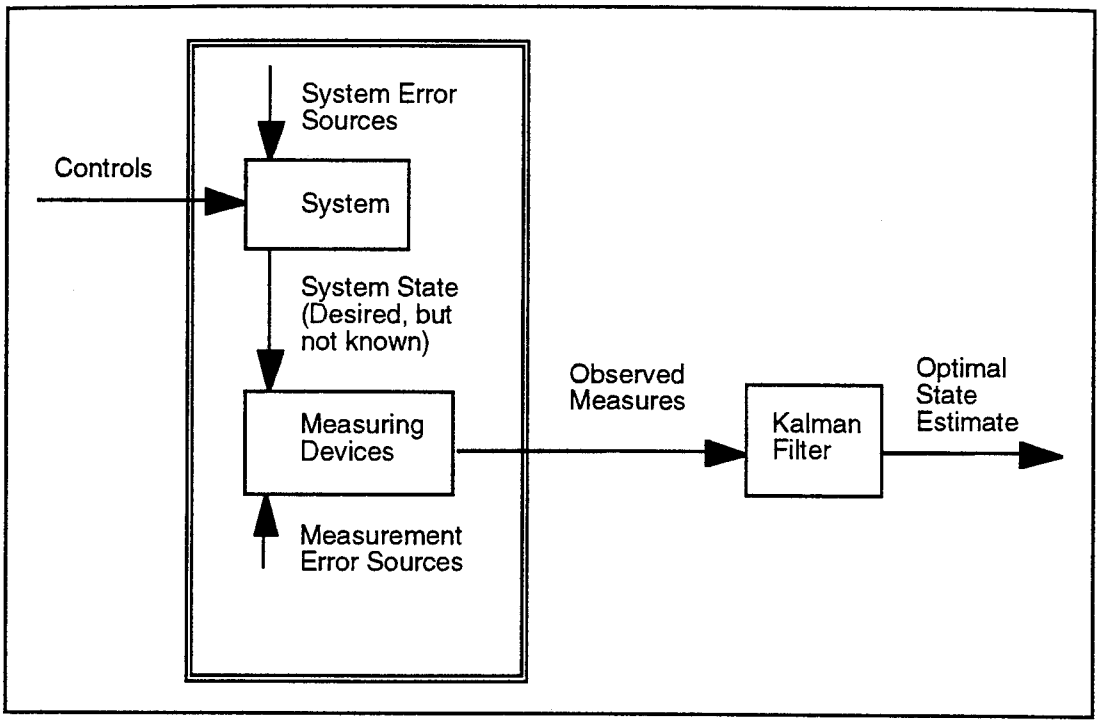


Figure 6: A Typical Kalman Filter Application

To compensate for errors and lack of information, we add a Kalman filter to our system, as shown in the right half of Figure 6. The filter takes as its inputs the observed (noise-corrupted and incomplete) measurements from the system measuring devices and uses them to generate an optimal estimate of the actual system state. It then propagates that estimate forward in time until the next measurements are available, when it repeats the process to produce another, updated estimate, and the cycle begins again.

In order to produce its estimates, a Kalman filter maintains state information in a number of vectors and matrices. These are briefly described below:

- $\hat{x}$ is the estimate produced by the Kalman filter. Typically, both an $\hat{x}^-$ (the predicted estimate) and $\hat{x}^+$ (the corrected estimate) are maintained.
- P is the filter-computed state estimate error covariance. Again, both a P^- from the prediction cycle and a P^+ from the correction cycle are maintained.
- Q is the modeled dynamics noise strength
- R is the modeled measurement noise covariance
- Φ (Phi) is the modeled state transition matrix
- B is the modeled control input matrix
- G is the modeled dynamics noise input matrix
- H is the modeled measurement matrix

The $\hat{x}$ vector and P matrix are updated during the predictor/corrector cycle according to the formulae given below. During the prediction cycle, the following propagations are made:

$$\hat{x}\left(t_i^-\right) = \Phi\hat{x}\left(t_{i-1}^+\right) + Bu\left(t_{i-1}\right) \tag{1}$$

$$P\left(t_i^-\right) = \Phi P\left(t_{i-1}^+\right)\Phi^T + GQG^T \tag{2}$$

where $\hat{x}\left(t_i^-\right)$ is the $\hat{x}^-$ estimate for time t_i, u is a control input vector for the Kalman filter, and the other symbols are as defined above. When the actual measurement data arrives, the correction cycle performs the following updates:

$$\hat{x}\left(t_i^+\right) = \hat{x}\left(t_i^-\right) + K(t_i)\left[z(t_i) - H\hat{x}\left(t_i^-\right)\right] \tag{3}$$

$$P\left(t_i^+\right) = P\left(t_i^-\right) - K(t_i)HP\left(t_i^-\right) \tag{4}$$

where $K(t_i)$ is the optimal gain matrix for time t_i. It denotes the relative weight used in the filter correction step, and is calculated as shown in Equation (5), below:

$$K(t_i) = P\left(t_i^-\right)H^T\left[HP\left(t_i^-\right)H^T + R\right]^{-1} \tag{5}$$

The other symbols are as defined previously.

To simplify the design and implementation of Kalman filters, we make several assumptions about the system environment:

- the dynamics of the system and the measuring devices can be adequately modeled by linear functions.
- the system noises (error sources) and measurement corruption noises are both white and gaussian with zero mean.
- we can identify an initial condition for the state probability density function.

These assumptions may at first seem too restrictive to make a Kalman filter practical, but closer examination shows that this is not the case.

The first assumption (linear characteristics) is justifiable for practical, historical reasons. A linear model is not only simpler to implement, but is often adequate to model system behavior. If the model has some nonlinear characteristics, engineers will usually use a linear approximation with linear perturbation techniques such as Taylor series truncated to first order terms about a nominal operating condition.

The second assumption indicates that noise sources are white and gaussian. White noise often confuses people. Essentially, "white" means that the noise value is not correlated in time; or in other words, knowing the value of the noise at time (t_{i-1}), or even at all times $(t_{i-1}, t_{i-2}, t_{i-3}, \ldots, t_0)$, tells you nothing about what the value will be at time t_i. Whiteness also implies that the noise has equal power per unit frequency (or power spectral density value) at all frequencies, which results in a noise with infinite power. Since this cannot exist in nature, this seems to destroy the usefulness of this assumption. However, any system has a system bandpass: that finite range of frequencies to which it will respond. Since this system bandpass is finite, we can approximate white noise by a power spectral density (PSD) value that is constant over a band of frequencies equal to (or slightly wider than) the system bandpass, and that decreases to zero outside the system bandpass. This approximated white noise has the desirable property that it has a finite power and therefore can exist in nature; we call it wideband noise. Figure 7 shows the relationship between bandpass, wideband noise, and white noise. From the point of view of the system's response, the wideband noise and true white noise are identical. Therefore, we can treat the wideband noise as white noise in our system noise model, which makes the noise model math more tractable. Also, we can again run the (fictitious) white noise through a linear system model called a *shaping filter* to produce noise with different power spectral density values at different frequencies, thus enabling the generation of a wide range of noise functions. Augmenting the shaping filter model to the original linear model yields an overall model in the form of a linear system driven by white noise.

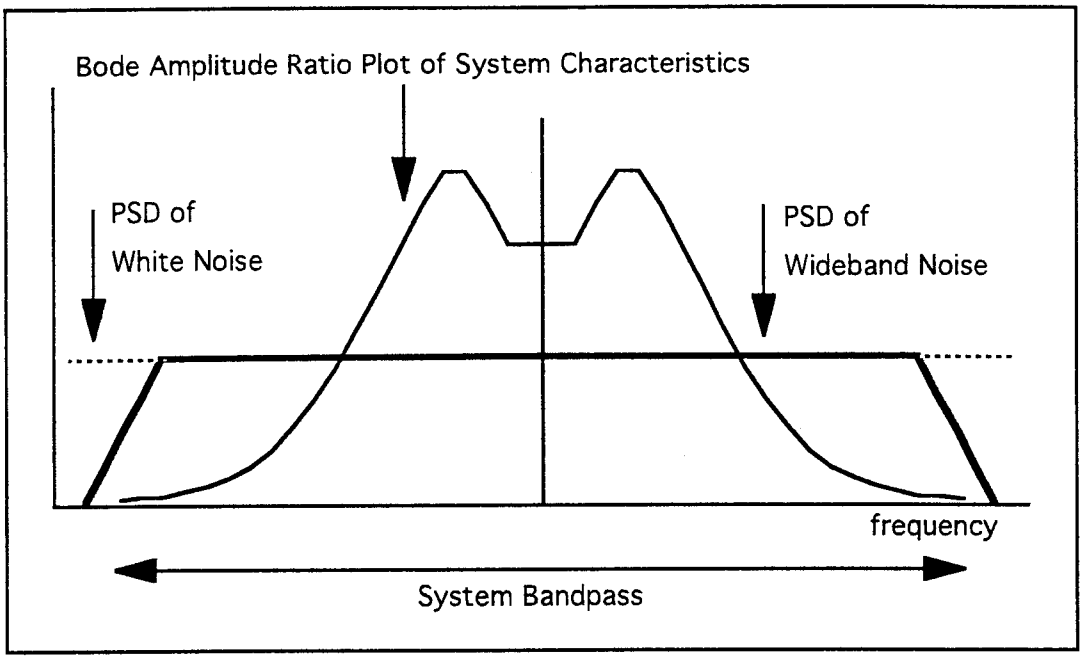


Figure 7: Bandpass, Wideband Noise, and White Noise

Whereas white describes the time correlation characteristics (or lack thereof) of a noise, gaussian describes its amplitude characteristics. A gaussian noise has the property that, at any given time, the probability density function of the noise has a gaussian shape. This can be justified physically by remembering that overall system noise is typically a summation of many small error sources; the Central Limit Theorem indicates that this summation is more and more gaussian as the number of sources increases. In fact, it can be shown that, for as few as three uniformly distributed (very nongaussian) variables, the probability distribution of the summation is approximately gaussian. Thus, it seems reasonable to assume that the noise in our system also follows this gaussian nature.

Gaussian distributions also have the desirable property of remaining gaussian when propagated by linear systems. Thus, the conditional density for the system state vector, conditioned on observed measurements, remains gaussian at all times. Also, gaussian distributions can be completely

characterized by a mean μ and a variance σ^2; this means that the filter can propagate *all* the information available to the current time by computing these two values each cycle.

The final reason for using gaussian distributions explains why Maybeck describes Kalman filters as "optimal, recursive, data processing algorithms" [Maybeck79:4]. Many different criteria exist for the word optimal, and this leads to different optimal estimates: the mode, or value with the greatest probability; the mean, or "center of mass" value; the median, that value such that half of the probability is to one side of it and the other half to the other; the midrange, or average of the smallest possible x value and the highest; and there are others. All of these definitions are equally valid, and for gaussian distributions all are the same value: the center of the distribution. So by any reasonable criteria, a Kalman filter working under the assumptions above will produce the *optimal* estimate of the system state.

The third assumption asserts that we incorporate into our filter any knowledge we have of initial system conditions. This allows the filter to generate an initial probability density distribution and initiate the prediction/correction cycle.

Kalman Filter Variations. There are a variety of Kalman filter models available for the system designer. Chang and Tabaczynski have written a survey article on the design and analysis of Kalman filters in which they describe several models useful in target tracking. The following discussion summarizes their article [Chang84:99-101].

Kalman filter approaches are differentiated by how they answer the following two questions:

- What is the assumed dynamics model for the target?
- How is K, the filter gain, computed?

The first question asks us to make an assumption about the type of motion that the filter will be expected to track. The two possibilities offered are Constant Velocity (CV) or Constant Acceleration (CA) models. CV models use a six-element state vector (three position variables and three velocity variables in a 3D problem) while CA models use a nine-element state vector (three positions, three velocities, and three accelerations). CA models (nine-element state vectors) are viewed as more appropriate for tracking a maneuvering target.

The second question relates to how the filter gain (K) is computed. As mentioned previously, K is a weight applied during the filter correction cycle. If K is very small, then the filter becomes insensitive to incoming measurements; if K is very large, then the filter tends to ignore past measurements. Four approaches to dealing with filter gain computation are presented: The extended Kalman filter (EKF); the finite memory filter; the fading memory filter; and the constant gain filter (CGF).

The *extended Kalman filter* (EKF) calculates the gain using Equation (5) above, but allows for nonlinear dynamics or measurement models to be employed in filter design. The *finite memory filter* calculates gain based only on the n most recent data points. The *fading memory filter* uses all measurements, but weights the result in favor of more recent measurements. This type of filter is also referred to as an *aging filter*. The final alternative is the *constant gain filter* (CGF). This approach is used when it is not possible or desirable to compute the filter gain in real time. Instead of computing a new gain each cycle, either a set of pre-computed gains or a single constant

gain is used. This method is the least expensive computationally (no gain computation is required).

Chang and Tabaczynski feel that EKF filters offer a good compromise between computational cost and filter performance for many applications. Constant Gain filters were seen as a viable alternative to the computationally more expensive extended Kalman filter if the performance degradation incurred could be tolerated. The other two approaches were not seen as viable alternatives due to computational costs incurred without corresponding performance gains. It should be noted that their evaluation considered only single-filter designs, and not MMAEs at all.

Thesis Approach. This research uses a nine-state model, but represents acceleration as a first-order Gauss-Markov process rather than employing the more simplistic constant acceleration proposed by Chang and Tabaczynski [Chang84]. Each of the elemental filters used is a constant gain filter (CGF) in order to take advantage of the low computational cost (in terms of both number of computations performed and time required).

2.3 Multiple-Model Adaptive Estimators (MMAEs)

A Kalman filter alone is not sufficient to deal with the lag problem. This is because a single Kalman filter cannot accommodate all the possible types of head motion that a typical virtual environment participant will exhibit. Recall that a Kalman filter is designed for a specific system operating condition. If we design a Kalman filter to generate head orientation estimates under the assumption that the participant is making only slow, benign head movements, we can expect that filter performance will be degraded if the participant moves his/her head in a manner different from

the hypothesis (rapidly, for example). We would therefore like to enhance our Kalman filter to account for this possibility, and produce a system that will not only generate optimal head orientation estimates, but react appropriately to changes in the participant's head motion characteristics. One approach to achieving this is through the use of a Multiple Model Adaptive Estimator (MMAE).

MMAE Basics. An illustration of an MMAE is shown in Figure 8. An MMAE uses K Kalman filters running in parallel, each of which is designed for a different hypothesized condition. In the case of head motion tracking, we might use three filters: one for benign, slow motion; one for moderate motion; and one for very rapid motion (more correctly a reacquisition motion, such as might occur when trying to keep track of more than one target in an environment).

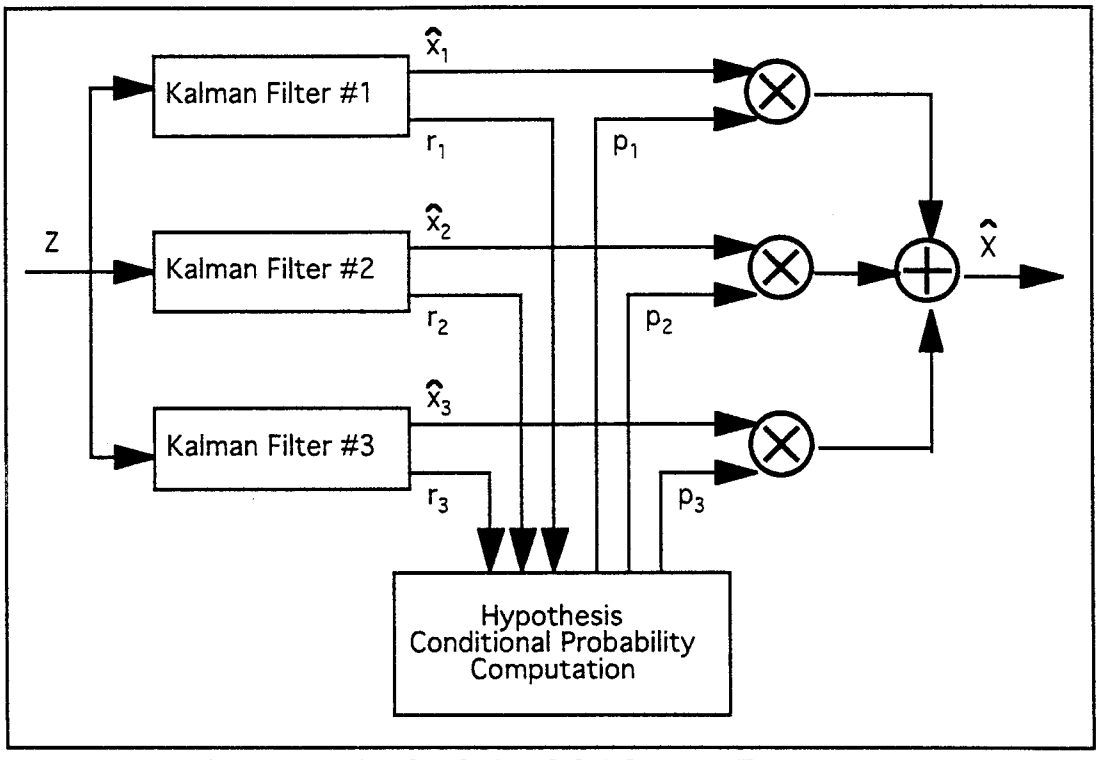


Figure 8: Multiple Model Adaptive Estimator

Since each of the Kalman filters in the MMAE will produce a state estimate, we need either a selection scheme to determine which is the best estimate, or a blending scheme to allow for in-between cases [Friedman92:60].

In the case of the selection scheme, each filter in the MMAE produces a state estimate, $\hat{x}_k$, and a residual, r_k $(1 \leq k \leq K)$, which is the difference between the actual measurement z at time t_i and the filter-predicted measurement (from the kth filter) before that actual measurement arrives. These residuals become inputs to a Hypothesis Conditional Probability Computation engine, which assigns a probability weight, p_k, to each filter based on the current residual and the previous weight. The probability weight p_k produced is the conditional probability that the associated Kalman filter has the *correct* hypothesis about the real world, conditioned on all measurements observed to that time. The filter with the smallest residual relative to that filter's computed residual covariance (and therefore the largest probability weight) is selected as the best-fit hypothesis and the state estimate from that filter becomes the MMAE estimate [Friedman92:61]. This is known as the *maximum a posteriori*, or MAP, version of the MMAE algorithm. To relate this to Figure 8, above, the filter with the greatest p_k weight as determined by the Hypothesis Conditional Probability Computation would be assigned a weight of 1.0; the other filters would be assigned a weight of 0.0 (this will be modified slightly due to concern for "zero lock-on", to be discussed later in this section).

The blending scheme works similarly except that the probability weights, once produced, are multiplied as-is by the appropriate filter estimate and the results summed to produce a final, optimal system state estimate as

a probability-weighted average of all the individual filter state estimates. Thus, a conditional mean estimate is produced rather than a conditional mode (as in the MAP form); this is often called the Bayesian version of the MMAE algorithm. Intuitively, the filter with the correct hypothesis should consistently produce the smallest residuals (relative to the corresponding filter-computed residual covariance). This causes the probability weight for that filter to grow closer to one, while the weightings for the other filters grow closer to zero. Thus, the final state estimate is weighted most heavily towards the estimate produced by the filter with the correct hypothesis. The formulae are given below:

$$p_k(t_i) = \frac{f_{z(t_i)|a,Z(t_{i-1})}\left(z_i|a_k, Z_{i-1}\right) * p_k(t_{i-1})}{\sum_{j=1}^{K} f_{z(t_i)|a,Z(t_{i-1})}\left(z_i|a_j, Z_{i-1}\right) * p_j(t_{i-1})} \tag{6}$$

where $f_{z(t_i)|a,Z(t_{i-1})}\left(z_i|a_k, Z_{i-1}\right)$ is a probability density function defined by

$$f_{z(t_i)|a,Z(t_{i-1})}\left(z_i|a_k, Z_{i-1}\right) = \frac{1}{(2\pi)^{\frac{m}{2}}\left|A_k(t_i)\right|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2} r_k^T(t_i) A^{-1}(t_i) r_k(t_i)\right\} \tag{7}$$

and the residual $r_k(t_i)$ and filter-computed residual covariance $A_k(t_i)$ are:

$$r_k(t_i) = z_i - H_k(t_i)\hat{x}_k\left(t_i^-\right) \tag{8}$$

$$A_k(t_i) = H_k(t_i) P_k\left(t_i^-\right) H_k^T(t_i) + R_k(t_i) \tag{9}$$

There are several practical concerns in MMAE design relevant to this research. One of these is the number of independent filters in the system. While it is theoretically possible to design an MMAE with a Kalman filter for every possible hypothesized system condition, it is not usually practical (or possible) to implement such a system. Therefore, it is incumbent upon the

designer to consider carefully the hypothesized system conditions when designing an MMAE. Appropriate discretization of the parameter space is essential to MMAE performance. Overly fine discretization (too large a choice of K) is computationally burdensome and can generate difficulties in discerning the difference in properties of residuals in different filters. Too coarse a discretization can yield the condition in which no filters have an assumed parameter value close enough to the true value to produce good estimation or small residuals.

Another concern is huge residuals. When a filter has a wrong hypothesis, we expect that the residual values for that filter will become very large, possibly too large for the computer system to represent. When this happens, the filter is said to be divergent. To prevent this, the MMAE monitors these residuals and, if any becomes too large, the appropriate filter is re-initialized with the current estimate $\hat{x}$ (emanating from the right side of Figure 8, once the effect of the divergent filter is removed).

A last concern is zero lock-on. Zero lock-on occurs when the probability weighting associated with a particular filter is allowed to go to zero, as due to huge residuals in a divergent filter. When this occurs, the weighting for that filter as computed by the iterative computation of Equation (6) will "lock-on" and remain at zero. This will result in outputs from that filter being ignored, even if the associated hypothesis should later become correct due to a change in the real world environment. There are several methods for dealing with this problem. One is to tune all filters to prevent initially incorrect filters from drifting too far from the true state. This has the associated drawback of causing all filters to look alike and be weighted essentially equally, incapacitating the adaptation process. Another common method is to enforce

an ad-hoc lower bound on the probability weightings, thereby preventing any filter from reaching a zero probability weighting.

Thesis Approach. For this research, a computationally tractable three-filter MMAE was chosen. The first filter was designed for slow, benign head movements; the second for moderate head movements; and the third for rapid movements such as occur in a target re-acquisition. The MMAE checked each filter after every measurement correction to ensure that the residuals for that filter had not become too large. If this occurred, the filter was re-started using the current MMAE estimate. Also, a lower probability bound of approximately 0.01 was enforced to prevent zero lock-on.

2.4 Software Architectures

With the advent of Software and Systems Engineering, the focus of producing software systems has moved somewhat away from data structures and algorithms and turned to the organization of the various components that comprise the overall system – the *software architecture*. There are tremendous benefits to be gained by more optimal arrangement of system components, more standard interfaces between components, and more effective means of describing and analyzing system components.

Historical Development. The history of software development has been marked by a desire on the part of practitioners to increase the level of abstraction; to move farther from the details of the hardware that they work with. In the 1950s, programming was done in machine language, and programs and data had to be explicitly placed in the computer's memory. Programmers soon realized, however, that certain sequences of instructions were being used and re-used quite frequently, and that replacing them with

simple, easy-to-remember character codes would reduce programming time and errors. This led to the first abstraction: substitution of mnemonic codes for sequences of machine instructions. This trend toward higher level abstractions continued with the development of more abstract programming languages to represent the new concepts and ideas being developed and make them simple for the programmer to use.

Today, we have again pushed the level of abstraction at which we work upward with the advent of software architectures and domains. In a draft version of Air Force Pamphlet 63-115, a software architecture is defined as "a top-level description of a software design defined early in the system's life cycle". It goes on to list the key component of a software architecture:

- Components. These are the building blocks of the system. They may be partitioned according to algorithmic functions, reusable components, associated objects, or any appropriate scheme.
- Relationships. These are the connections between components, and define the data and control interfaces both between the components themselves, and possibly between the software system and the outside world. These relationships should also allow for analysis of the architecture to determine such things as critical timing paths and throughput attributes.
- Style. These are the guidelines and principles to be used in implementing the relationships defined above, and also any constraints which must be recognized. [USAF93:4-41,4-42]

There are many benefits to designing a system around an existing software architecture. First, and perhaps most important, is the time savings involved. Designing new systems as variations of similar existing systems (and reusing the existing architecture and design information) greatly

reduces the required design time. Second, reusing an architecture also allows the designer to reuse the knowledge and experience of the original designer. Finally, a good software architecture allows the designer, developer, and potential customers to visualize complex systems and relationships in a more natural, intuitive fashion.

General Architectural Styles. David Garlan and Mary Shaw have described many current architectural styles such as filters and pipes; data abstraction and Object-Oriented organization; event-based, implicit invocation; layered systems; repositories; interpreters; and heterogeneous systems that are combinations of the aforementioned styles [Garlan93:4-13]. Except where noted otherwise, the material for the following discussion is based on material from their article.

In *pipe and filter* systems, each component, or *filter*, takes in a set of well-ordered inputs, does some local transformation on them, and incrementally produces a set of well-ordered outputs. The connectors between filters are called *pipes*. The filters in a pipe and filter system are constrained to work in a vacuum, without knowledge of what their input or output sources are. This architectural style is often used in compiler and distributed system design.

Data abstraction and Object-Oriented (OO) organization groups system entities and their allowable operations together as objects. Objects communicate and interact with each other through function and procedure invocations. As with the pipe and filter systems above, each object in this style should be independent, neither required to have information about any other object in the system, nor required to request needed information for determination of its own state from any other object in the system.

Event-Based systems move away from the more traditional styles defined above. Instead of waiting passively for an outside entity to request a service, agents in an event-based system *broadcast* (or *announce*) the occurrence of an event. When this happens, all other agents in the system that have declared an interest in that event are notified. Agents declare an interest in a particular event by having a procedure or function call for that event that is invoked by the system manager when the event occurs. Thus, the event broadcast *implicitly* invokes the corresponding procedure or function calls. Note that an agent cannot guarantee the order in which other agents will respond to an event, or know when they finish responding. This style is commonly used in packet-switched networks, user interfaces that separate data presentation and data management, and some programming environments.

Layered systems are hierarchical systems in which each layer provides more (and usually more abstract) services than the layer beneath it. Usually, each layer is accessible only by the layers surrounding it in the hierarchy. Protocols define interaction between layers. This style of architecture is most commonly used in defining communications systems (such as the International Standards Organization Open Systems Interconnect, ISO OSI, model).

A *repository* can be broken up into two components: a central data store that maintains the current state of the repository; and one or more independent components that operate on the central data store. One common example of a repository is a *blackboard architecture* in which the *blackboard* is a shared common data base that is accessed and updated by independent *knowledge sources* containing application-dependent information. Blackboard

systems have been used to implement speech and pattern recognition systems.

A *Table-driven Interpreter* is a virtual machine implemented as software. Interpreters take a pseudo-program as input, and produce an execution of the program as output. As an example, the BASIC language is usually an interpreted language. Programs written in BASIC are not generally compiled into an executable; rather they are interpreted by a BASIC execution engine, and the results returned to the user. Interpreters are commonly used to overcome differences between the user's expectations of a system, and the actual abilities of the hardware.

One other architectural style mentioned by Garlan and Shaw is the *domain-specific architecture*. Domain-specific architectures seek to define a generic reference framework for a specific software domain (such as command and control or database management systems) that can then be specialized for specific system instances [Garlan93:13]. Much recent work has focused on defining software domains and developing architectures for them. Don Batory and Sean O'Malley, however, have developed a domain-independent architectural *meta-model*. Their meta-model allows complex system architectures to be built up from low-level *components* belonging to one or more *realms*. Members of a realm have the same interface; thus members of a realm are *plug-compatible* [Batory91:2-3].

Much recent work in this area has been done at AFIT. Mark Snyder developed ObjectSim, an object-oriented simulation architecture that allows graphics applications to create and manipulate scenes [Snyder93]. Snyder's work is being continued and expanded by Jordan Kayloe, who is translating the ObjectSim code into Ada 9X and also expanding its capabilities

[Kayloe94]. Work also continues to bring software engineering and Ada 9X into the graphics research domain at AFIT.

The Object Modeling Technique (OMT). OMT is an object-oriented approach to the analysis and design of software systems that was developed by Dr. James Rumbaugh and his colleagues at the General Electric Research and Development Center. OMT focuses on identifying elements of the system (called *objects*) and defining their behaviors and the relationships between them. This is in contrast to more traditional functional decomposition techniques that focus on specifying and decomposing system functionality [Rumbaugh91:6]. OMT is a very rich modeling technique that covers many aspects of software specification and design. However, this introduction will focus on those aspects of OMT that are used in this research.

One of the models available in OMT is the *Rumbaugh diagram*. This model shows the objects in a system, as well as their relationship to other system objects. A simple Rumbaugh diagram is shown in Figure 9. Objects in a Rumbaugh diagram are enclosed in boxes. These boxes may optionally

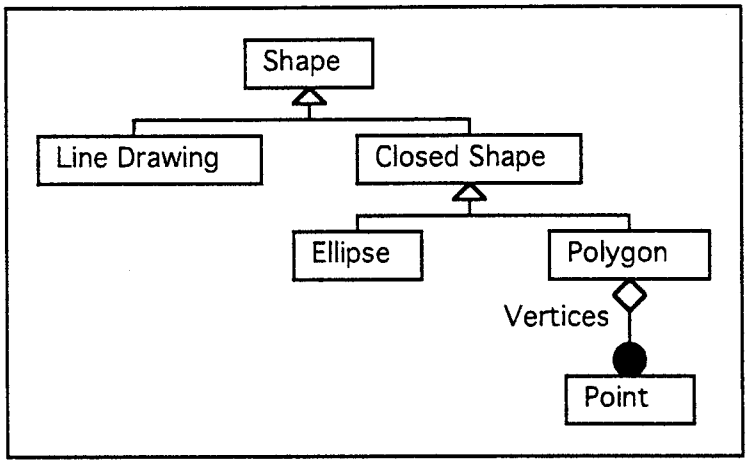


Figure 9: Rumbaugh Diagram for a Shape Class
[Adapted from Rumbaugh91:44]

have three subdivisions: one for each of object name, object attributes (usually state variables), and object operations (allowable operations on the object). Object can also be related to other objects, and OMT provides two relationships. The first, represented by a triangle, is the inheritance relationship (also known as the "is-a" relationship). An object connected underneath another object by this relationship inherits all the attributes and operations of the higher-level object. An example of this relationship in Figure 9 is the relationship between Shape and Closed Shape. Another way of viewing this is by specialization/generalization; we say that Closed Shape is a specialization of Shape, or conversely that Shape is a generalization of Closed Shape. The same can be said of Shape and Line Drawing. In terms of class relationships, Shape is a super-class, and Line Drawing and Closed Shape are sub-classes of Shape.

The other pre-defined relationship in Rumbaugh diagrams is the aggregation relationship, denoted by a diamond symbol. This relationship indicates that an object includes, or is made up of, other objects. This is demonstrated in Figure 9 by the relationship between Polygon and Point. A Polygon object includes Point objects in its definition; hence the relationship. Aggregation relationships can also have *multiplicity balls* that denote the cardinality of the relationship. An open circle indicates zero or more instances of the contained object (the contained object is optional) while a filled circle indicates one or more instances. Additionally, a number or range of numbers (such as 2+ for "at least two", or 1..4 for "one to four, inclusive") may be used to be more specific about the multiplicity.

Other relationships are denoted by a simple line (with possible multiplicity balls) between two objects. When this general relationship

symbol is used, a label that describes the relationship is usually added to the symbol. An example of a labeled relationship is shown between Polygon and Point in Figure 9. In this case, the relationship is that a point is a vertex of the polygon that contains it. Please note that labels can be added to the pre-defined relationships as well.

Thesis Approach. An object-oriented architectural approach was selected for this thesis work. Object-oriented design and programming techniques are used both at AFIT and abroad, and offer the designer many advantages. Object-oriented techniques help the designer to focus on the problem domain, thus producing systems that are "based on the underlying framework of the application domain itself, rather than the ad-hoc functional requirements of a single problem" [Rumbaugh91:6]. Focus on the domain leads to designs that are more amenable to later requirements changes; not as prone to the massive *ripple effects* of requirements changes noted in systems designed through functional decomposition.

A graphically-based modeling technique was desired for doing the system design work. Graphical representations are often simpler to understand that corresponding text-based descriptions, and also often convey information in a way that is more intuitive. The Object Modeling Technique (OMT) is such a graphical tool, and its rich notation and modeling power are more than adequate for this research.

2.5 Summary

In order to do this research, a large amount of background information was needed. This information came from the virtual environments, Kalman filter, MMAE, and software architectures areas.

Virtual environments are not a new concept, but rather the most recent step in a continuing development that started in the 1960s with Ivan Sutherland. During the 1970s and into the 1980s, research split into several different areas, but it is now beginning to reconverge as technologies developed in one area meld with technologies from other areas.

Kalman filters have also been around for quite a while, and have been used successfully in tracking-related applications. There are many different types of Kalman filter, each of which offers the designer a different set of abilities and constraints. Chang and Tabacynski's survey article indicates that the best overall filter design is the Extended Kalman Filter, or EKF. They feel it strikes a good balance between computational load and filter performance. The Constant Gain Filter is also seen as a good choice. It offers lower computational load and somewhat poorer overall performance, but it is still a good filter design.

MMAEs seek to enhance Kalman filter design by using two or more filters in parallel to adapt to changes in the characteristics of the task being performed. In terms of this research, MMAEs allow a predictor to adapt to changes in the head movement patterns of a virtual environment participant. Two MMAE designs were discussed; the Maximum A Posteriori, or MAP, design; and the Bayesian design. The MAP design selects the filter with the best prediction (as determined by the filter residual, or difference between the filter prediction and the actual measured data) and uses that filter's output as the MMAE output. The Bayesian design weights each filter according to its residuals, then multiplies the filter estimate by its associated weight and sums the results to produce an estimate that is a weighted mean of the filter outputs.

Finally, software architectures seek to describe systems as a collection of well-defined components. This is a relatively young field, and much research is currently underway to identify various architectures and domains of similar applications. Several current architectural styles were mentioned including object-oriented design, which seeks to model a system as a collection of independent but interacting objects. Each of these objects has its own state and behavior. This particular architecture/design style is the style that was used in this research.

III Software Design and Implementation

The purpose of this chapter is to provide the reader with the thought process used and decisions made during the design and implementation of the software. Good documentation of the requirements and design process is vital to the software engineering process. Good documentation not only facilitates discussion between the original designer and the user, but also provides a means for subsequent designers and implementors to make similarly informed decisions about modifications and/or extensions to the system.

Unless otherwise noted, the software engineering definition of the word *state* is used throughout this chapter. In software engineering, state is defined in reference to object behavior. In the words of Rumbaugh:

> *A* state *is an abstraction of the attribute values and links of an object. Sets of values are grouped together into a state according to properties that affect the gross behavior of the object.* [Rumbaugh91:87]

Essentially, a state represents a set of possible variable values for which the response of the object to stimuli (such as method calls from other objects) is identical. This definition makes it possible to discuss object behavior at a qualitative level; the detail of quantifying variable values is abstracted away. This also reduces the number of object states, thus simplifying graphical representations.

Commonly, an object includes variables for the express purpose of identifying the object's state. These are referred to in this document as *state variables*. An example of this is the FOGMA_Filter class, which includes the state variables filter_initialized and filter_divergent. The values of these

variables have a direct role in determining the state of the FOGMA_Filter object.

3.1 Prototype Work

In order to gain a better understanding of the technical aspects of the application to be developed, a similar existing virtual environment application called *redflag* was studied and enhanced. Redflag, which was developed by Major Michael Gardner [Gardner93], used position and orientation data collected from actual aircraft during training exercises to generate a virtual environment that allowed a participant to replay the mission. A Polhemus Isotrak sensor attached to an HMD worn by the participant allowed the view into the environment to change as the participant changed head orientation. A prototype software MMAE had been added to this software by captains Doug Blake and Bill O'Connor [O'Connor92].

Initial testing by the author showed that the application did not meet performance expectations. Subsequent analysis uncovered serious problems that needed to be addressed before the software could be used. A partial listing of these software problems, and the actions taken to correct them, is contained in Appendix A of this document. Generally, the problems fall into three main software engineering areas:

- Object Encapsulation. The prototype code did not encapsulate the various objects it used. Instead, a more ad-hoc method of adding code where it was most convenient seems to have been used. As an example, matrix operations, Kalman filter operations (for three filters), and MMAE operations were all implemented in a single monolithic

object. This made isolation and/or correction of errors very difficult and time-consuming.

- Testing Strategy. Several of the errors discovered in the prototype code were related to bad values (either nonsense values or values that were outside the prescribed range for the variable) being assigned to critical state variables. In most instances, these errors were readily apparent when the values of these variables were printed on a routine basis. The conclusion here is that proper testing techniques for these variables, and the algorithms used to update them, were not used. Also, these errors were not detected because the code did not contain range checks for input values.

- Unwise re-use. The prototype software re-used several objects from earlier work as though they were library objects. However, these objects did not have stable behavior. Therefore, the behavior of the prototype software was also unstable.

3.2 System Requirements and Constraints

In order to use the software developed in this research as a basis for further work, it was necessary to define the requirements and constraints under which the software is expected to operate. This will allow future designers to make informed decisions concerning the ability or inability of this software to meet their needs.

In defining the requirements for this software, the author relied on both his own experience with the prototype code and the requirements developed by Shaw [Shaw93] for a virtual environment application. The requirements and constraints are discussed below.

Frame Rate. The system must operate at a *reliable* minimum of 10 Hz (ten frames shown to the participant per second). Shaw points out that the

pivotal requirements for virtual environment applications are high frame rate and low system lag [Shaw93:291-292]. He also states that systems with frame rates less than ten Hertz are not considered responsive [Shaw93:292]. Also, most current applications in the AFIT graphics laboratory run at between 12 and 15 Hz, so 10 Hertz seems a reasonable choice for this research.

Performance Data. The system must provide the designer with a method to gather data that will allow analysis of system performance. This will allow the designer and developer to characterize the overall system performance, and to develop metrics for comparing various implementations.

Shaw points out that currently available performance measures are not always helpful. Many allow measurement of CPU time, but not actual time; and actual time is also of interest when designing a virtual environment (a process that is waiting for I/O is just as bad as one that is taking too long to do a computation) [Shaw93:292]. However, some measure of performance is desirable, and CPU time can serve as an indicator even though it may not be all-inclusive. This situation is somewhat analogous to the argument that lines of code (LOC) measures, while not a true measure of software quality, can be an indicator of aspects of software quality.

Object Encapsulation. The objects in the system must be encapsulated so that change in the implementation of a particular system object do not necessitate corresponding change in other system objects (the "ripple effect"). This also supports the notion of plug-compatible components by enforcing high cohesion within objects and loose coupling between objects. Also, testing, debugging, and enhancing the software will be made easier.

Portability. The software developed for this research must be portable. Historically, AFIT has undergone an upgrade to the computer systems (either in the computer platforms themselves, or in software support packages and libraries) approximately every two years; this has invariably led to problems with platform- or support-specific software in the application base. Such dependencies are not inherently undesirable; indeed, often the rationale for choosing one platform or support package over another is that it provides features (not present in the other) that are advantageous to the work at hand. However, these dependencies should be kept to a minimum and clearly identified so that portability impacts can be properly determined.

Readability and Understandability. These are always concerns for the software engineer, but in this case they are even more paramount. It is unlikely that follow-on research will be done by software engineers; more likely, this line of research will be continued by electrical engineers. Since no assumptions can be made about the level of software engineering experience on the part of these researchers, and since this research is to form a foundation for them to build upon, it is very important that this software be designed and implemented in as intuitive a fashion as possible. This will allow future researchers to leverage their productivity by reusing the design and implementation knowledge and experience embodied by this research.

One of the methods used to improve readability and understandability within the software was a set of naming conventions. Essentially, constants used within the software are in all capitals to make them easily recognizable (example, PATH_SIZE). Since there is no way to distinguish individual words, underscores are used to separate words in compound names. Type names and function names all begin with a capital letter, and also capitalize

the first letter of each word in compound names (example, FilterWeight). Variables are done in all lower case letters (example, viewpoint). There is an exception to this convention; abbreviations (example, MMAE) are typed in all caps regardless of what is being named. Also, matrices are named with a capital letter (example, Xc) to follow mathematical convention.

3.3 System (Architectural) Design

In order to meet the requirements and constraints, the application needed to interact with several external entities. These interactions were captured using the Rumbaugh Object Modeling Technique (OMT) [Rumbaugh91], and are presented as Figure 10.

Essentially, the application needs to interact with: a tracker, which in the case of this research communicates across an RS-232 (serial) communications port; and two display devices (the screen, and a Head-Mounted Display unit, or HMD).

The application was then broken down into a class diagram, also modeled with OMT. This diagram is shown as Figure 11. Class details shown in the figure are discussed later in this chapter.

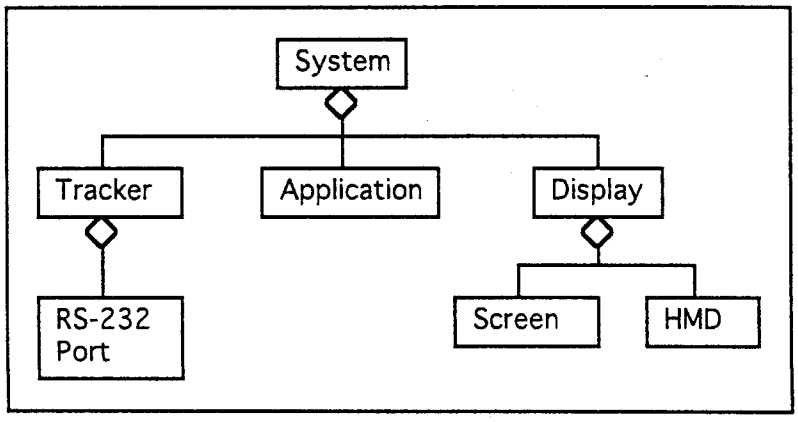


Figure 10: System-Level Architecture

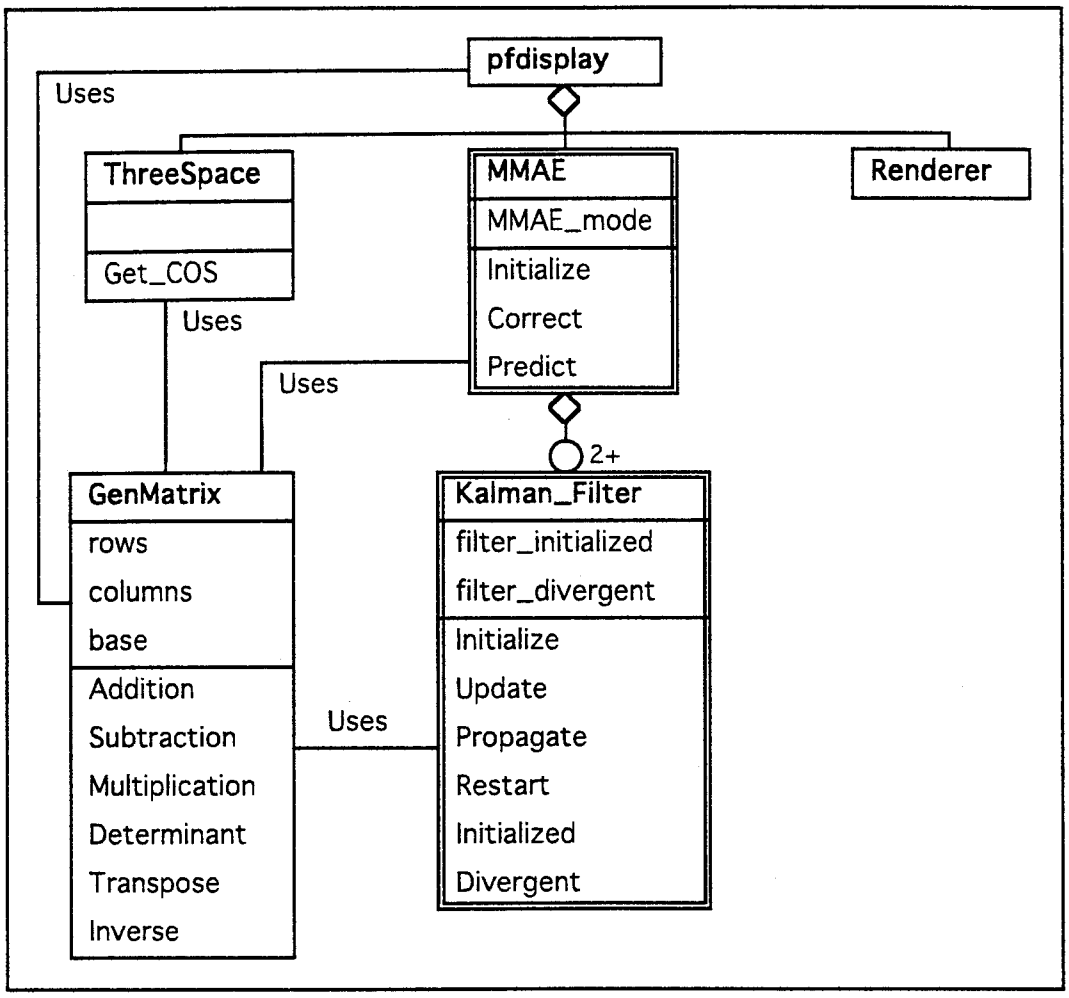


Figure 11: Top-Level Class Diagram

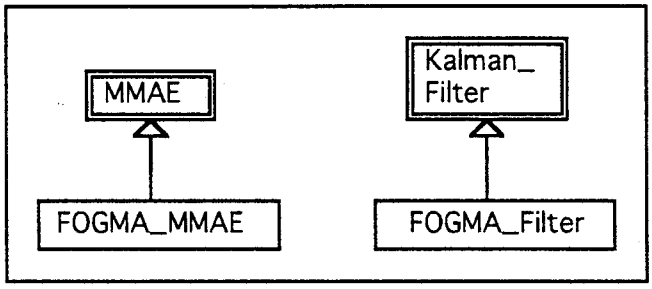


Figure 12: Concrete Implementations for Abstract Classes

In Figure 11, MMAE and Kalman_Filter are abstract classes; they are the foundation for a group of related classes. This research provided only one such related class for each, shown in Figure 12.

The ThreeSpace class is the interface to the Polhemus 3SPACE magnetic tracker. The name is taken from Polhemus 3SPACE magnetic tracker, and was chosen to highlight that this is **not** intended to be a general purpose class. This is discussed in more detail in a later section. The renderer class needs are met by the Performer [Performer90] rendering library.

MMAE and Kalman_Filter were implemented as abstract classes for two reasons. First and foremost, this approach encapsulates those data structures and operations that are common to *any* such class implementations. This provides future developers with a template from which to develop other concrete implementations.

Secondly, any class derived from the abstract class is automatically considered to be of the same type as the base class, and will inherit all of its data structures and method definitions. In other words, the First-Order Gauss-Markov acceleration filter class, FOGMA_Filter, is considered to be of type Kalman_Filter, and will inherit all of the data structures and method definitions in Kalman_Filter; this is also true for any other class derived from Kalman_Filter. This property of the derived classes supports Batory and O'Malley's idea of plug-compatible realms of software components [Batory91]. A realm contains a number of components, each of which presents the same interface to the external world. As long as that interface requirement is met, then software components from the same realm can, with

some limitations, be interchanged without affecting the operation of the calling software.

3.4 GenMatrix Class

MMAEs and Kalman filters are inherently dependent on matrices and vectors. Both MMAEs and Kalman filters use matrices and vectors to maintain state information. Further, state changes normally involve a large number of matrix operations. While these operations could be performed on an array (or set of arrays), the intimate relationship between MMAE behavior and matrix operations made development of a separate matrix class that localized and encapsulated the matrix operations an appropriate and obvious choice. Unfortunately, the existing matrix classes at AFIT were designed for graphics applications, and could support only square matrices of dimensionality three or four. Therefore, a general matrix class, GenMatrix, was developed to support the matrix operations needed for the thesis software.

GenMatrix implements a matrix as a dynamically allocated two-dimensional array; the dimensions of the array are provided by the client program when the matrix is instantiated. The matrix elements are stored as floating point numbers. It would have been possible to implement GenMatrix in such a way that any numeric type could be used as a matrix element, but this was not done. Recall that one of the requirements for this research was readability and understandability of the code. Implementing GenMatrix for a generic element type would be more complex (and therefore less readable and maintainable) than the float version. Also, floating point representations are

appropriate for a wide range of applications, including those defined in this research.

GenMatrix supports most of the commonly used matrix operations (addition, subtraction, multiplication, determinant, inverse, and transpose), as well as methods to add or multiply each matrix element by a constant. GenMatrix also includes methods to clear the matrix (set all matrix elements to 0.0), return the matrix dimensionality (the number of rows or columns), set or return the value of an individual matrix element, and print the matrix elements.

GenMatrix operations, to the maximum extent possible, verify matrix compatibility before performing the operation. As an example, the multiply method verifies that the matrices have the same inner dimension (if A is an m x n matrix, then B must be n x p in order to multiply A by B). If the matrix dimensions are not compatible for the operation requested, then the operation is not done. How the calling procedure is notified of this decision is dependent on the implementation, and is discussed later in this chapter.

3.5 Kalman Filter Design

With the GenMatrix class complete, the next class developed was FOGMA_Filter. As mentioned previously, FOGMA_Filter was designed as a derived sub-class of Kalman_Filter. Kalman_Filter defines the minimal, common interface for *any* Kalman filter implementation, and includes the following methods:

- Initialize – Initializes the filter.

- Update – Updates the state of the filter for an actual measurement, Z, which is an input to the method.
- Propagate – Generates and returns a prediction of the filter state one sample period into the future.
- Restart – Resets a filter that has become divergent to a known state.
- Initialized – Indicates whether or not the filter has been successfully initialized.
- Divergent – Indicates whether or not the filter is currently divergent.

In addition to the methods described above, two state variables, filter_initialized and filter_divergent, are included in the Kalman_Filter class definition.

The behavior of Kalman_Filter is as follows. When instantiated, the Kalman_Filter object is in the Uninitialized state, and will accept only an Initialize or Initialized method invocation (Initialized will indicate that the filter is not initialized). When Initialize is invoked, the object is initialized, and it transitions to the Operational state. In this state, the object will accept and respond to method invocations as long as it is not divergent. If it should become divergent, it transitions to the Divergent state. In this state, it will accept only a Restart or Divergent method invocation (Divergent will indicate that the filter is divergent). When Restart is invoked, the filter will reset and then transition back to the Operational state. A diagram of this behavior is included as Figure 13. Note that in the figure, *Method Call* indicates an invocation of any of Propagate, Update, Initialized, or Divergent. It is used as a shorthand notation to keep the diagram from becoming cluttered.

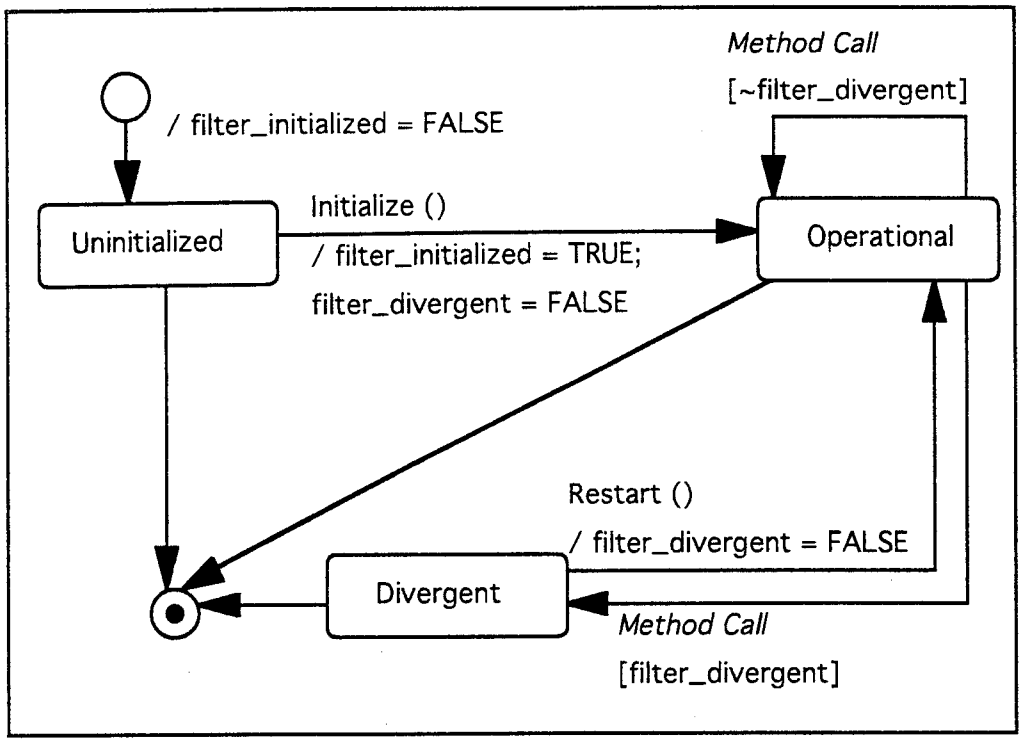


Figure 13: Behavior Diagram for Kalman_Filter Class

FOGMA_Filter Software Design. FOGMA_Filter models a Kalman filter that uses a First-Order Gauss Markov acceleration (FOGMA) dynamics model to generate its predictions. The most important method added to FOGMA_Filter is Project (as in "to throw ahead"). This method allows the filter to generate a prediction (or projection) of its state n sample periods into the future, where n is an integer greater than or equal to two. Project was added to the FOGMA_Filter design because the prototype code indicated a need for it. In the prototype code, the time required to read data from the Polhemus Isotrak was equal to the time needed to generate a frame for display (1/10th of a second). Therefore, a prediction of two sample periods into the future (to overcome both the lag in the data read and the lag in frame generation) seemed necessary.

The other methods added to FOGMA_Filter all provide the client with visibility to the various state variables. Methods are included to return the following values:

- RAR (the likelihood quotient). This is an indication of the confidence level of the filter estimate. The formula for RAR is given below:

$$RAR = r^T A^{-1} r$$

where r is the residual vector (the difference between the filter predicted measurement and the actual measurement for the same time) defined as

$$r = z - H\hat{x}\left(t_i^-\right)$$

and A is the filter-computed covariance for the residual defined by

$$A = HP\left(t_i^-\right)H^T + R$$

The equations for r and A actually appear earlier in this document as part of Equations (3) and (5), respectively.

- RR (the ME/I likelihood quotient). This is similar to RAR, defined above. The formula is

$$RR = r^T r$$

where r is as defined above. The difference between RAR and RR is that RAR is scaled by the A term whereas RR is not.

- $\hat{x}^-$, $\hat{x}^+$, P^-, P^+, as defined in Section 2.2.

- K (the filter gain). The gain is an optimal weighting value used as a correction to the filter. Very large gain values indicate high relative weight on incoming measurements, and low relative weight on the filter output.

- the current residual vector (the difference between the predicted filter state, and the actual data value for the same sample period); the formula for this is shown above.

Two additional points are worth mentioning. First, the filters used in this research were tuned under the assumption that inputs (in the form of actual measurements) are constrained to be in the range [-1.0, 1.0]. This is because the Polhemus 3SPACE magnetic tracker returns directional cosines, which are always in that range. If, however, a different input source is desired, the filters will have to be re-tuned for the new input ranges.

The second point concerns values for filter initialization. In order that the filter's operating characteristics might be changed without having to recompile the code, the values needed for initialization were stored as ASCII text files. These files are read in by the filter initialization method, and contain the following values in the order specified:

- H, R, P^-, $\hat{x}^-$, as defined in Section 2.2.
- σ (sigma), the standard deviation of the head acceleration assumed by the filter.
- τ (tau), the correlation time of the head acceleration assumed by the filter.
- T, the sample period (the inverse of the sample rate, or number of times per second that the filter is to generate a prediction).
- RAR_Limit (the minimum RAR value at which the filter is considered to be divergent).

FOGMA Filter Technical Design. The First-Order Gauss-Markov acceleration (FOGMA) model was chosen because it is simple, linear, and has

been used successfully in many tracking environments. It also has desirable properties in terms of on-line computational load. A Kalman filter normally requires initial values for eight variables ($\hat{x}^-$, P^-, Q, R, Φ, B, G, and H) in order to initialize (these variables are described in Section 2.2). The FOGMA model allows the values of Φ, Q, and P^- to be pre-computed off-line if suitable values can be determined for the standard deviation σ and the correlation time τ for the head motion acceleration assumed by each of the filters.

The particular implementation used in this research also had another desirable property. The filters do not expect or allow for any control inputs from the participant; therefore, we can set B [the control input matrix in Equation (1)] to zero and G [the modeled dynamics noise input matrix in Equation (2)] to the identity matrix. This in turn allows the prediction equations defined in Section 2.2 [Equations (1) and (2)] to be simplified. Since B = 0, the second term in Equation (1) can be eliminated; further, since G = I, it can be eliminated from Equation (2). The resulting equations are shown below, and were implemented in the FOGMA_Filter:

$$\hat{x}\left(t_i^-\right) = \Phi\hat{x}\left(t_{i-1}^+\right) \tag{10}$$

$$P\left(t_i^-\right) = \Phi P\left(t_{i-1}^+\right)\Phi^T + Q \tag{11}$$

All terms in Equations (10) and (11) are as defined previously. The correction equations cannot be further simplified, so remain as defined in Section 2.2, Equations (3) and (4). The Project method uses Equation (10), but takes the current Φ and multiplies it by itself n-1 times (remembering n is an integer input greater than or equal to two) before multiplying the result by the

current (this is appropriate in the case of a time-invariant model with constant Φ).

H, R, $\hat{x}^-$, and T were identical for all three filters. The measurement matrix H is shown below. It was determined from the fact that only position measurements (indicated by the 1 entries in the first three columns) are available, and the fact that $\hat{x}^-$ is comprised of three position values, three velocity values, and three acceleration values (in that order):

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The measurement noise covariance R was determined through trial and error because the Polhemus 3SPACE User's Manual [Polhemus90] does not provide any direct values for sensor noise characteristics. This omission is understandable since noise values are dependent on environmental conditions. In fact, Steve Bryson describes Polhemus sensor noise as "significant, very noisy at distances of greater than 45 to 50 inches", and "very sensitive to location" [Bryson92:254]. Experimentation in the AFIT graphics laboratory showed acceptable filter performance at assumed noise variance levels of 0.01 with approximately three feet separating the source and sensor. This value was used in the R matrix, resulting in the following:

$$R = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

It seemed reasonable to assume that the participant would initially be looking straight ahead into the virtual environment, and that his/her head would (at least initially) be still. In other words, the initial velocity and acceleration components of the $\hat{x}^-$ vector would be zero, and the position

components would indicate "straight ahead". In the case of the thesis environment, "straight ahead" is defined by a right-hand coordinate system with the participant looking along the positive Y-axis, the positive X-axis to the right, and the positive Z-axis straight up. The equivalent directional cosine values are (0, 1, 0). This leads to the initial value for $\hat{x}^-$ shown below:

$$\hat{x}^- = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Finally, the sample period, T, for the filters was set at 1/10th of a second (0.1 seconds) to correspond to the frame rate (10 Hz) used in the application.

Φ, Q, and P^- were determined for each filter based on that filter's σ and τ values. σ is the standard deviation of the head motion acceleration assumed by the filter; it represents the bandwidth to which the filter will respond. τ is the correlation time of the head motion acceleration assumed by the filter; it indicates how long a given head motion acceleration persist in time and is inversely proportional to the assumed bandwidth of the acceleration process. The formulae for the elements of Φ and Q were based on the work of Tobin [Tobin86]. The formula for the Φ matrix is as follows:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 & f_{15} & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & f_{15} & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & f_{15} \\ 0 & 0 & 0 & 1 & 0 & 0 & f_{35} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & f_{35} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & f_{35} \\ 0 & 0 & 0 & 0 & 0 & 0 & f_{55} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & f_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & f_{55} \end{bmatrix}$$

where

$$f_{15} = \tau\left[T - \tau\left(1 - \exp\left(\frac{-T}{\tau}\right)\right)\right] \tag{13}$$

$$f_{35} = \tau\left[1 - \exp\left(\frac{-T}{\tau}\right)\right] \tag{14}$$

$$f_{55} = \exp\left(\frac{-T}{\tau}\right) \tag{15}$$

and T is the sample period. Equations (13) through (15) are from Tobin's work [Tobin86:52].

Similarly, the Q matrix is also based on Tobin's work, and is defined by

$$Q = \begin{bmatrix} q_{11} & 0 & 0 & q_{13} & 0 & 0 & q_{15} & 0 & 0 \\ 0 & q_{11} & 0 & 0 & q_{13} & 0 & 0 & q_{15} & 0 \\ 0 & 0 & q_{11} & 0 & 0 & q_{13} & 0 & 0 & q_{15} \\ q_{13} & 0 & 0 & q_{33} & 0 & 0 & q_{35} & 0 & 0 \\ 0 & q_{13} & 0 & 0 & q_{33} & 0 & 0 & q_{35} & 0 \\ 0 & 0 & q_{13} & 0 & 0 & q_{33} & 0 & 0 & q_{35} \\ q_{15} & 0 & 0 & q_{35} & 0 & 0 & q_{55} & 0 & 0 \\ 0 & q_{15} & 0 & 0 & q_{35} & 0 & 0 & q_{55} & 0 \\ 0 & 0 & q_{15} & 0 & 0 & q_{35} & 0 & 0 & q_{55} \end{bmatrix}$$

where

$$q_{11} = \sigma^2\left\{\frac{2\tau T^3}{3} - 2\tau^2T^2 - 4\left[\tau^3T * \exp\left(-T/\tau\right)\right] + 2\tau^3T - \tau^4\exp\left(-2T/\tau\right) + \tau^4\right\} \tag{16}$$

$$q_{13} = \sigma^2\left\{\begin{array}{l}\tau T^2 + 2\left[\tau^2T * \exp\left(-T/\tau\right)\right] + \tau^3 - 2\tau^3\exp\left(-T/\tau\right) - 2\tau^2T + \\ \tau^3\exp\left(-2T/\tau\right)\end{array}\right\} \tag{17}$$

$$q_{15} = \sigma^2\left\{-2\tau T * \exp\left(-T/\tau\right) + \tau^2 - \tau^2\exp\left(-2T/\tau\right)\right\} \tag{18}$$

$$q_{33} = \sigma^2\left\{2\tau T - 3\tau^2 + 4\tau^2\exp\left(-T/\tau\right) - \tau^2\exp\left(-2T/\tau\right)\right\} \tag{19}$$

$$q_{35} = \sigma^2\left\{\tau - 2\tau\exp\left(-T/\tau\right) + \tau\exp\left(-2T/\tau\right)\right\} \tag{20}$$

$$q_{55} = \sigma^2\left\{1 - \exp\left(-2T/\tau\right)\right\} \tag{21}$$

Again, the formulae are taken from Tobin's work [Tobin86:53-54].

The final initialization matrix, P^-, was generated as the appropriate steady state solution to Equations (1), (4), and (5) from the given Φ, H, Q, and R matrices using Matlab [MatLab90]. An example code segment is included below:

```
Phi = [Appropriate Phi matrix]
I = [1 0 0 0 0 0 0 0 0;
     0 1 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0;
     0 0 0 1 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0;
     0 0 0 0 0 1 0 0 0;
     0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 1]
H = [1 0 0 0 0 0 0 0 0;
     0 1 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0]
Q = [Appropriate Q matrix]
R = [ 0.01 0    0;
      0    0.01 0;
      0    0    0.01]
[K, Pm, Pp, E] = dlqe (Phi, I, H, Q, R)
save KF.out K Pm Pp -ascii
```

The **d**iscrete **l**inear **q**uadratic **e**stimator (dlqe) function provides a Kalman filter based solution for a given Φ, G, H, Q, and R. It returns the steady state P^-, P^+, and K matrices. In our case, G = I (the identity matrix), so a 9x9 identity matrix is used in the code segment.

The Φ and Q matrices were generated by the filter as part of the initialization process. However, since the P^- matrix was generated by Matlab, it was read in as part of the filter data file.

3.6 MMAE Design

Similar to FOGMA_Filter (and for the same reasons), FOGMA_MMAE was implemented as a derived sub-class of the abstract class MMAE, which defines the minimal data structures and methods for any MMAE application. MMAE defines the following methods:

- Initialize – Initializes the MMAE.
- Correct – Updates the MMAE state for an actual measurement, Z, which is an input to the method.
- Predict – Generates and returns a prediction of the MMAE state one sample period into the future.

Additionally, the state variable MMAE_mode is defined. Please note that no Initialized or Divergent method is defined for the MMAE class.

The behavior of the MMAE class is also similar to that of the Kalman_Filter class. When instantiated, the MMAE object is in the Uninitialized state, and will only respond to an Initialize method invocation. When Initialize is invoked, the MMAE initializes and transitions to the Operational state. While in this state, method invocations will be accepted and processed as long as at least one filter in the MMAE is not divergent. If all filters within the MMAE become divergent at the same time, then the MMAE transitions to the Divergent state. This state is more a notational convenience for the diagrams than an actual state, because the MMAE cannot return to the Operational state; it will immediately transition from the Divergent state to the Dead state. The Divergent state simply allows an opportunity to print a warning message or take clean-up actions. The behavior diagram is shown as Figure 14:

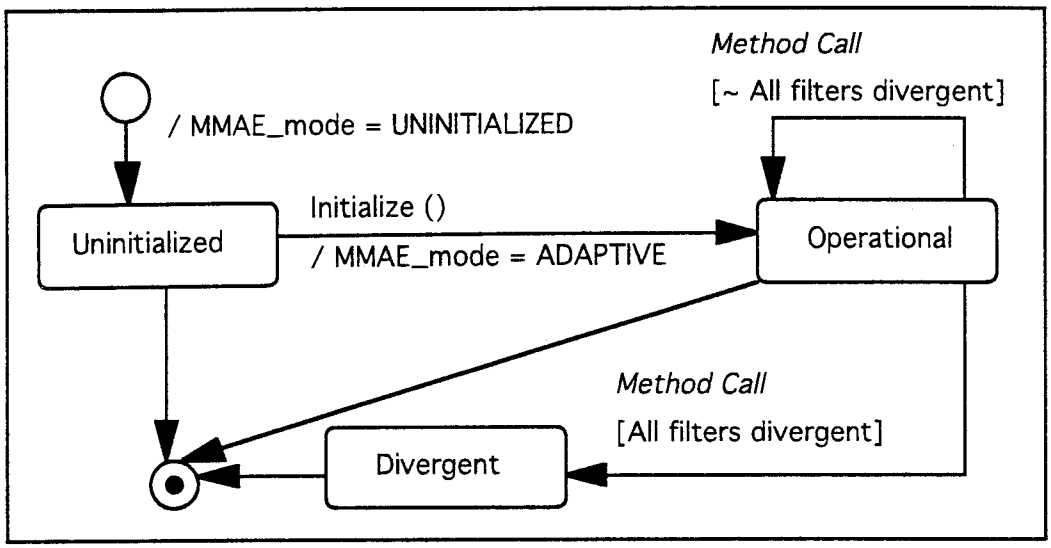


Figure 14: Behavior Diagram for MMAE Class

Here, *Method Call* indicates an invocation of Predict or Correct.

FOGMA_MMAE Software Design. FOGMA_MMAE models an MMAE that comprises up to ten FOGMA_Filters. Objects of this class are instantiated with a number of filters, and a lower bound for the probability value associated with a particular filter (as discussed in Section 2.3).

FOGMA_MMAE extends the MMAE class by including several new methods, and new behavior. Essentially, it was necessary to allow the MMAE to operate as though it had only a single filter in order to study the performance of the individual filters and to *tune* each of them (tuning a filter involves adjusting its σ and τ values for best filter performance). To implement this, a new method, SingleFilterMode, was added. This method takes an integer input n in the range [0, K], where K is the number of filters in the MMAE. If $n = 0$, then the MMAE will use the outputs from all filters in its calculations. If $n > 0$, then the MMAE will only use the outputs from filter n for its calculations. This modification required a complementary

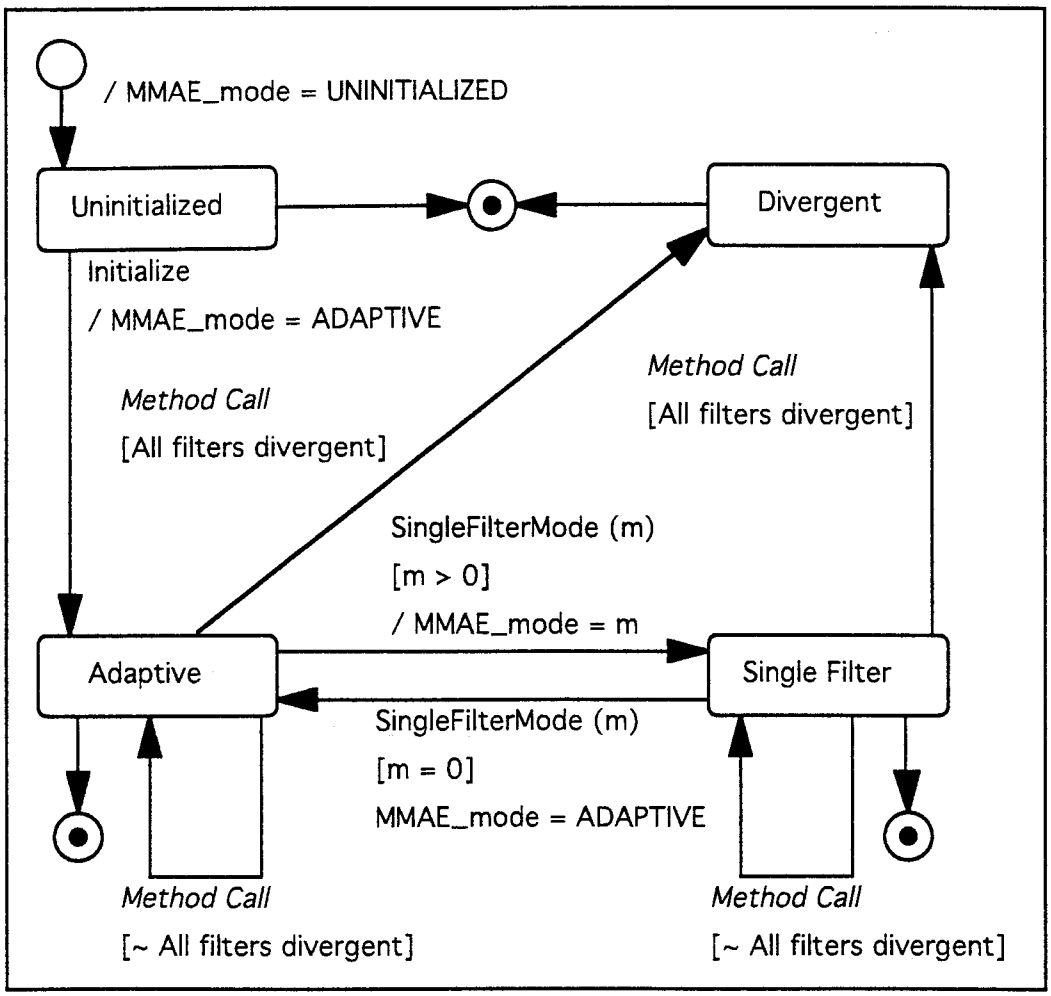


Figure 15: Behavior Diagram for FOGMA_MMAE Class

change in the behavior diagram for the class. The new behavior diagram is shown in Figure 15:Note that the figure contains more than one Dead state. This was done to keep the diagram from becoming too cluttered.

Accommodating a Single Filter mode also required changing the way that estimates ($\hat{X}$ values) are calculated. As explained previously, the MMAE normally generates a probability weight for each filter based on all filters' residual values (the difference between the filter-predicted values and the actual data values for the same time). These probability weights are then multiplied by the filter estimates, and the results summed to produce a final

estimate for the MMAE. However, in Single Filter mode, this behavior is not desired. Instead, the probability weight for that filter should be set to one, while all other filters should get probability weights of zero. This same philosophy carries over to the FilterWeight method, which returns the current probability weight for a particular filter.

In addition to SingleFilterMode and FilterWeight, several other methods are included. There is an Extend method that mirrors the Project method of FOGMA_Filter; it allows the MMAE to generate a prediction of its state more than one sample period into the future by using the current Φ and probability weights as constants. There are also mirror methods for all of the methods to retrieve the value of various state variables. The difference is that FOGMA_MMAE prepends each method name with *Filter* (FilterXHatMinus, FilterXHatPlus, and so on), and requires the client to specify the filter (in the range [1, K]) for which values should be returned.

FOGMA_MMAE Technical design. Three filters were used in the application. Filter #1 was designed under the hypothesis that the participant is making benign head movements. An example of a benign movement is a person in an automobile tracking the path of the car directly ahead. The motion of the car is (normally) fairly predictable, and not subject to sudden changes in direction or speed. Filter #2 used the hypothesis that the participant was looking around, tracking a moderately to harshly moving single target. Continuing with the driving example, this type of motion might be exhibited by a driver trying to follow the motion of a car that has encountered a patch of ice on the road, and is now swerving back and forth somewhat wildly. This motion was classified as moderate. Filter #3 used the hypothesis that the participant was making large, rapid swings of the head,

such as occur in a reacquisition task. An example is when a driver hears a horn from behind, and did not know a car was there, so quickly turns to find out where the car is. The various values for σ and τ are shown in Table 1:

Filter	sigma	tau
1 (Benign)	0.06	1.0
2 (Moderate)	1.0	0.2
3 (Heavy)	5.0	0.05

Table 1: Sigma and Tau Values for MMAE Filters

Initial values for these variables were taken from the prototype code. Then the filter performance was examined by the researchers. To do this, a researcher would hold the Polhemus 3SPACE sensor and make movements of the appropriate type. The individual filter outputs and the interplay between filters would then be examined, and adjustments to the σ and τ values made if needed. This trial and error process was repeated until satisfactory filter performance was achieved.

3.7 ThreeSpace Class

The final class object developed was a ThreeSpace class that provided an interface between the application and the Polhemus 3SPACE magnetic tracker that provided the head orientation estimates. This code was largely based on an existing Fastrak class developed by Capt Mark Gerken [Gerken91] and later modified by LtCol Philip Amburn. This existing interface was deemed unsuitable for use without modification for the following reasons:

- The code read more information than was needed for the application, and did not provide a means to specify what information should be read.
- The read time was unacceptably long (on the order of one hundred milliseconds).
- Testing showed that it would not properly support the Polhemus 3SPACE tracker provided by Armstrong Laboratories.

In order to correct these problems, a new class, ThreeSpace, was developed. However, since the goal of this research was to develop an MMAE, not a ThreeSpace, an engineering decision was made not to attempt to make this a general class. The resulting design is, therefore, very specific to the needs of the application and the requirements of the 3SPACE tracker, and contains many hard-coded assumptions (such as the port number to which the 3SPACE is attached, and the baud rate of the port). The ThreeSpace class supports only one method, Get_COS, that reads the cosine values from the Polhemus 3SPACE magnetic tracker.

3.8 C++ Implementation

Because most of the graphics applications at AFIT use C++ as the implementation language, and because the Performer [Performer90] software library is written in C, the first version of the application was developed in C++. C++ offers many advantages to the developer. It is a very terse language; it doesn't require the developer to learn a great deal of syntax. It is also a very good prototyping language. Most importantly to this research, it is an object-oriented language, and includes support for class constructors and destructors, polymorphism, and inheritance.

During the C++ implementation, several new problems with the prototype code were uncovered, as well as some flaws in the original design. Since these discoveries and work-arounds also form part of the overall design decision process, they are included in this section. The various comments are divided up according to the object to which they apply.

GenMatrix Class. The initial version of GenMatrix had only one class constructor that initialized all GenMatrix elements to zero upon allocation. This was very useful since C++ does not clear memory on allocation. However, this caused later compilation problems when the FOGMA_Filter class attempted to declare members of type GenMatrix. This problem could be avoided by making the members in question pointers to type GenMatrix, but this was an unsatisfactory solution for two reasons:

- it required changing all the code in the FOGMA_Filter class to dereference the pointers before invoking any methods in GenMatrix; this made the FOGMA_Filter class code much less readable.
- it flew in the face of being able to use user-defined types as though they are native types, which is one of the goals of the C++ language.

After several days of struggling with the problem, the solution was found in the following passage from Stroustrup's book [Stroustrup91] that explained a *ctor initializer*.

> *"An object of class M can be a member of a class X only if (1) M does not have a constructor, or (2) M has a default constructor, or (3) X has a constructor and if every constructor of class X specifies a* ctor-initializer *... for that member."* [Stroustrup91: 579]

A ctor initializer is a special form of constructor that invokes other class constructors. After reading the passage, the GenMatrix class was changed to have a default constructor that does nothing, and the constructor for the

FOGMA_Filter class was changed to be a ctor initializer. This solved the compilation problems, and allowed the FOGMA_Filter code to use members of type GenMatrix.

To complement the new (null) default constructor, an Initialize method was added to the GenMatrix class. Functionally, a (null) GenMatrix declaration followed by an Initialize method invocation is equivalent to a (non-null) GenMatrix declaration alone. While this is somewhat redundant, it has the advantage of allowing the declaration of dynamically-sized arrays of GenMatrix objects. This does introduce the potential for uninitialized GenMatrix variables, however, if a user does not remember to invoke the Initialize method.

Another problem with GenMatrix occurred when trying to overload the brackets ([ ]) operator used to reference an individual matrix element. Two arguments (row and column offset) were needed, but the brackets operator will accept only zero or one argument(s). The author was not experienced enough in C++ to know how to use a recursive call and still perform adequate bounds checking on the inputs, so he developed a work-around. An Elt method that can be used on either side of an assignment statement was written to replace the brackets operator. This is not as elegant as a recursive brackets operator, but it is satisfactory and also provides some contextual indication of when a variable of type GenMatrix is used.

The GenMatrix class does not, in general, return status values to the calling program. Instead, if an operation cannot be completed due to bad input values or other conditions, an appropriate message is printed and program execution is halted via the Exit command. This is not an optimal engineering solution to the problem, since a subordinate object is making

decisions about program continuation for higher level objects, but it is the most reasonable course for two reasons:

- If the values sent to an operation do not allow the operation to be accomplished (for example, when trying to take the determinant of a singular matrix), then the results of that operation would be undefined. Returning a value that is not in any defined state was not seen as a viable implementation strategy.

- C++ does support exception handling through the Throw/Catch and Assert language extensions. Unfortunately, neither of these extensions is supported by the version of C++ used to implement the software. Therefore, there was no reasonable way to use exceptions to indicate when operations were unable to complete.

FOGMA_MMAE Class. Two modifications of the prototype design were made for this class. The first modification involved altering the constructor for the class to include a lower probability bound for the MMAE filters. This allows the user to determine what bound to use, and sets the same bound for all MMAE filters (using a single bound across all filters is standard practice in MMAE design).

The second modification was to rewrite the Correct, Predict, and Extend methods to take advantage of the parallelism of their execution. Essentially, all three methods call the appropriate FOGMA_Filter method to update the estimate variable (or a temporary copy in the case of Extend), and then generate a best estimate based on the current value of estimate and weight, taking into account whether or not the MMAE is currently in single filter mode. Taking advantage of this parallelism led to the redesign of these methods and the creation of a private method, WeightedEstimate, that encapsulates the code used to calculate the MMAE estimate.

ThreeSpace Class. Several versions of this class were implemented in C++ in order to find the best alternative among the design possibilities. The first version of the software decreased the size of the data record sent by the Polhemus by removing unneeded data fields. The Fastrak code returned the full set of nine direction cosines (three values for each of the three direction axes), as well as the 3D position of the sensor relative to the Polhemus source, even though the prototype code only used a subset of the cosine data, and totally ignored the position data. For the thesis code, only three cosine values were needed (the three X-Direction cosines), so the ThreeSpace class was designed to return only those. The result was an approximate doubling of the number of reads per second possible. Then the baud rate of the connection between the host computer and the Polhemus was doubled (from 9600 baud to 19200 baud) without changing any other system parameter. This resulted in a further increase in reads per second possible. Finally, the data format being sent from the Polhemus was changed from ASCII text to binary data; this again reduced the size of the data record being sent by the Polhemus, and also again increased the number of reads possible per second. A table showing the various configurations and the time required in milliseconds to perform a single read is shown in Table 2.

Version	Record Size (bytes)	Format	Baud Rate	Read Time (milliseconds)
Prototype	92	ASCII	9600	100
First	26	ASCII	9600	50
Second	26	ASCII	19200	38
Third	11	Binary	19200	27

Table 2: Read Times for Various Software Configurations Under IRIX 4

Note that the figures in Table 2 were taken with the software running on an SGI 4D platform operating under the IRIX4 operating system. Also, in order to make the binary read work, it was necessary to change the port configuration used by the software.

One would think from Table 2 that the binary read is the best choice, especially when the goal of keeping the read time as short as possible is taken into account. However, this is not the case. The ASCII reads all achieve 100% reliability; in other words, a good data record is returned each time the Polhemus is requested to send a data record. The binary format read could only achieve an approximate 60% reliability, and resulted in a visual display that jumped erratically due to bad data being returned. This drop in reliability may be due to the relative speeds of the processor and the Polhemus across the serial communications line; an exact reason was not established. However, the ASCII read at 19200 baud (Second version in Table 2) was chosen for the implementation since it had the lowest time per read that still maintained 100% reliability.

Main Application (pfdisplay). In order to achieve the desired visual effects (highlights on the balls, display of filter information and crosshairs), it was necessary to change the structure of the main application. Specifically, it was necessary to write call-back routines that were invoked by Performer, and that used shared data. The important point about this from a software engineering standpoint is to highlight the close coupling between the main application and the Performer library. The main application is structured in the way demanded by the Performer software, and if it should become desirable in the future to move away from Performer, then the main application will have to be rewritten to suit whatever software is used.

Also, the C++ implementation was developed under the IRIX 4 operating system. At the time of this writing, only one SGI platform in the graphics lab at AFIT supports IRIX 4; the rest have been upgraded to IRIX 5.2. These compatibility of these operating systems is limited, and the C++ software will have to be modified in order to port it to the new operating system.

3.9 Ada Implementation

Implementing the application in Ada 9X involved working under two constraints. The first was a desire to make the Ada 9X application as structurally comparable to the C++ already developed as possible. The motivation for this was to make comparisons between the two versions as unbiased as possible. However, meeting this goal was more difficult than anticipated, since C++ and Ada 9X implement object-oriented features in different ways. Several design changes were needed in order to translate the C++ into Ada 9X.

The most obvious of these changes is the use of Ada package specifications in place of C++ header files. The fundamental Ada unit is the package, and Ada objects are most naturally defined as packages. The interface to the outside world is defined by the package specification, while the implementation of the object behavior is hidden in the package body. It is worth noting that Ada package specifications and C++ header files are not completely analogous; package specifications are more tightly coupled to their respective bodies than header files to their respective implementation.

Another change involved type declarations. C++ allows you to define an object and then use the name of that object as a type name when declaring

other variables. Ada 9X takes a somewhat different approach. You declare a type within a package specification that is then available for outside objects to use in declarations.

The final design change points out the difference in philosophies between C++ and Ada 9X. C++ takes the viewpoint that user-defined classes should be able to behave in the same manner as native language types. To achieve this, almost everything in C++ is overloadable. This means that the developer has tremendous latitude in defining object behavior. However, it also places a great deal of responsibility on the developer to be thorough in designing new classes. Ada 9X has a somewhat different philosophy on user-defined types. The user is free to overload certain operations, but others are reserved for the language. While this may at first seem restrictive, it can actually decrease the workload on the designer, because the responsibility to provide definitions for reserved operations falls on the language developers.

An example of these philosophical differences can be seen in the implementation of the GenMatrix class. The C++ version contains methods for assignment, retrieval of individual elements, setting of individual elements, and other methods that do not appear in the Ada 9X implementation of the class. The reason for this discrepancy is that Ada 9X provides definitions of these methods that are generic enough to accommodate the GenMatrix class without requiring new definitions.

The second constraint was imposed by the compiler selected for the implementation. SGI provided a set of Ada bindings for Performer [Performer90] that allow Ada 9X to make the Performer calls necessary to generate the virtual environment used in the research. These bindings were compiled with the GNAT Ada 9X compiler; therefore, the same compiler was

used to compile the thesis software. The GNAT compiler is currently fairly immature, and does not yet fully support all language structures of Ada 9X. An example of one of these unsupported features is generics. The original design of the MMAE class called for it to be implemented as a generic. The reason a generic was chosen was that the number of filters used by the MMAE is not known at compile time. Implementation as a generic allowed the number to be passed to the object when it was instantiated. Unfortunately, the GNAT compiler had trouble compiling the generic properly, and the result was that the generic worked perfectly in one executable, but did not work at all in another. Finally, the MMAE class was re-designed to avoid use of generics.

Even working under these constraints, the Ada 9X implementation offers some advantages. Ada 9X allows the developer to constrain the ranges of the various elements with type definitions, thus precluding the need to check ranges of input variables. Such a feature in C++ may have helped the prototype developers to catch the initialization errors in their code. Ada 9X also supports the boolean type, thus avoiding the less elegant C++ integer implementation with TRUE and FALSE explicitly defined as integer 1 and 0, respectively. Perhaps the greatest advantage of Ada 9X, however, is its ability to define and handle exceptions. Exceptions allow the developer to differentiate between what is normal processing, and what is not. When combined with the strong type checking noted above, Ada 9X offers the developer a powerful method for ensuring program correctness. Also, by using exceptions, the decision of whether or not to terminate program execution can be left to the highest level object in the system. This is in direct contrast to the C++ implementation of this software, which has lower

level objects causing program termination because they have no way to inform the calling program that they cannot complete an operation.

3.10 Summary

In the ideal design and implementation environment, all aspects of the design would be equally important, and all aspects of the implementation would be meticulously validated and verified. However, software engineering, like other types of engineering, is not an ideal environment; trade-offs and compromises are inevitable. Software engineering allows the user to determine the importance of various aspects of the design and implementation, thus producing systems tailored to his/her needs. Put another way, the engineering process allows trade-offs to be made once the sensitivities of the system in question are identified. The role of the software engineer in this process is to ensure that, when trade-offs are made, they are made in an appropriate, informed manner. This philosophy was applied to the design and implementation of the thesis software. The prototype software provided a foundation from which to develop new requirements and designs. Those requirements then became the driving factor in subsequent design decisions.

Two versions of the thesis software were developed; one in C++, and one in Ada 9X. Although both of these languages are object-oriented, each of them has its own philosophy on how object features should be implemented. These differences in philosophy resulted in two noticeably different implementations, each with different strengths and liabilities. C++ offers the ability to integrate user-defined types almost seamlessly into the native language. This, combined with the ability to overload virtually any language

construct, results in a tremendous ability on the part of the developer to identify and define object behavior. This freedom, however, comes with a responsibility to thoroughly understand the consequences of the behaviors defined. C++ performs type conversions without informing the user in order to make operations compatible. The results of these conversions is not always well-defined. It is therefore the responsibility of the C++ practitioner to understand the environment, and ensure well-defined, reliable behavior. Ada 9X does not allow the developer the same freedoms as C++. It does offer many other amenities however. Strong type checking and the ability to constrain variables to specific values combine to foster a development atmosphere that prevents errors instead of responding to them. The ability to raise exceptions allows the developer to separate the normal processing of the system from the exceptional. However, Ada 9X does not yet have the mature compilers necessary for large-scale, commercial development.

IV Data Collection and Analysis

This chapter describes the various experiments performed in order to validate the application and the software engineering process. The application was validated both by testing performed during implementation, and by a performance study. The software engineering aspects of the research were studied and the C++ and Ada 9X implementations of the software were compared.

4.1 Implementation Testing

Testing during the implementation phase fell under three main areas. The first was the traditional unit-level testing that stresses the software in order to identify, isolate, and correct any coding errors or unstable behaviors. The next form of test was a filter tuning test. These tests studied the behavior of the Kalman filter objects both individually and together. The final form of test was a performance test that allowed the behavior of the Kalman filters and the MMAE to be baselined.

Test Programs. During implementation, several test programs were developed to ensure the proper functioning of the system objects. The first of these, *filter_test*, allowed testing of the FOGMA_Filter object. The user indicated which filter (corresponding to an assumed benign, moderate, or heavy head motion acceleration) to test, and also which path file (containing simulated Polhemus data) to use as input data. The program read in the path file and normalized the data values so that they approximated angular cosine data read from the Polhemus. The appropriate filter was then initialized, and the simulated Polhemus input used to drive the filter.

Program output could be directed to either the screen or an output file. A sample output of this program is included in Appendix B of this document.

The next program was *mmae_test*. This program performed essentially the same function as filter_test, but used the MMAE object. The user could specify which path file to use as simulated input. Test output could be directed either to the screen or to a data file. Additionally, the user could select to have some outputs directed to a chart file. The chart file used a comma-separated, ASCII text format to allow the results to be ported to a spreadsheet, chart, or word processing program. A sample output of this program (not including the chart file output) is included in Appendix B of this document.

The final program was *square_wave*. This program also tested the MMAE object, but allowed the user to specify an angular displacement (in degrees) and a duration (in frames, where one frame is equal to one tenth of a second) to use as the input generator for the test. Similar to mmae_test, above, the user could opt to have an ASCII text output file created. The main purpose of these test programs was to aid in identification, isolation, and correction of any errors in the software; however, they also became a useful tool for characterizing the performance of both the individual filters, and the MMAE. Systematic data could be collected with these programs much more reliably and repeatably than with actual Polhemus data collected from human subjects.

Filter Tuning. During initial coding of the filter software, each filter was tested with the filter_test program and an appropriate data set. Test outputs were examined for correctness (no mathematical errors in the formulae used to generate the outputs) and also for validity (reasonable

output values relative to the input values). Once the individual filters were running satisfactorily, the MMAE software was written, and the mmae_test program was used to check correctness and validity of the MMAE outputs. Once the MMAE was operational, the filters were tuned. Because no baselines existed, an empirical, trial and error method based on observed filter performance was used to tune the filters.

The first attempt at filter tuning looked at performance across all filters. With the MMAE running, a researcher used the Polhemus to generate an appropriate motion type (benign, moderate, or heavy), and the MMAE outputs (in terms of RAR values as defined in Section 3.5) and individual filter weights) were examined to see if they matched the input motion (the probability weight for the appropriate filter being assigned 0.98 or something reasonably close to that). Then, the motion type would be changed (from benign to moderate motion, for example), and the interplay between the filters examined. The goal of tuning in this fashion was to ensure that the correct filter was chosen for the motion type exhibited, and that when the motion characteristics changed, that the probability weight shifted to the new correct filter. If this was not the case, then the σ and τ values for the filters were adjusted, and the new configuration re-tested. This cycle was repeated until a tuning that provided satisfactory performance was found.

Individual filter performances were then examined. The MMAE was run in single filter mode, and the performance of the individual filters assessed in terms of overshoot and sluggishness. Again, if a filter's performance was not acceptable (too much overshoot observed, or too sluggish a return to a fixed position when motion was stopped) then the σ and τ

values for the filter were adjusted, and the filter re-tested. Eventually, a tuning that provided satisfactory performance for both the MMAE and the individual filters was reached. This tuning was used to generate baseline performance data.

Performance Characteristics. Filter RAR values (again, as defined in Section 3.5) are critical to the performance of the MMAE. RAR value indicates an inverse level of confidence in the filter prediction; the larger the RAR value, the less likely it is that the filter prediction is correct. The filters in this research tend to start losing confidence (and therefore giving up probability) at RAR values greater than approximately 4.0. Figure 16 shows the RAR values for each of the filters at angular displacements of zero through ninety degrees.

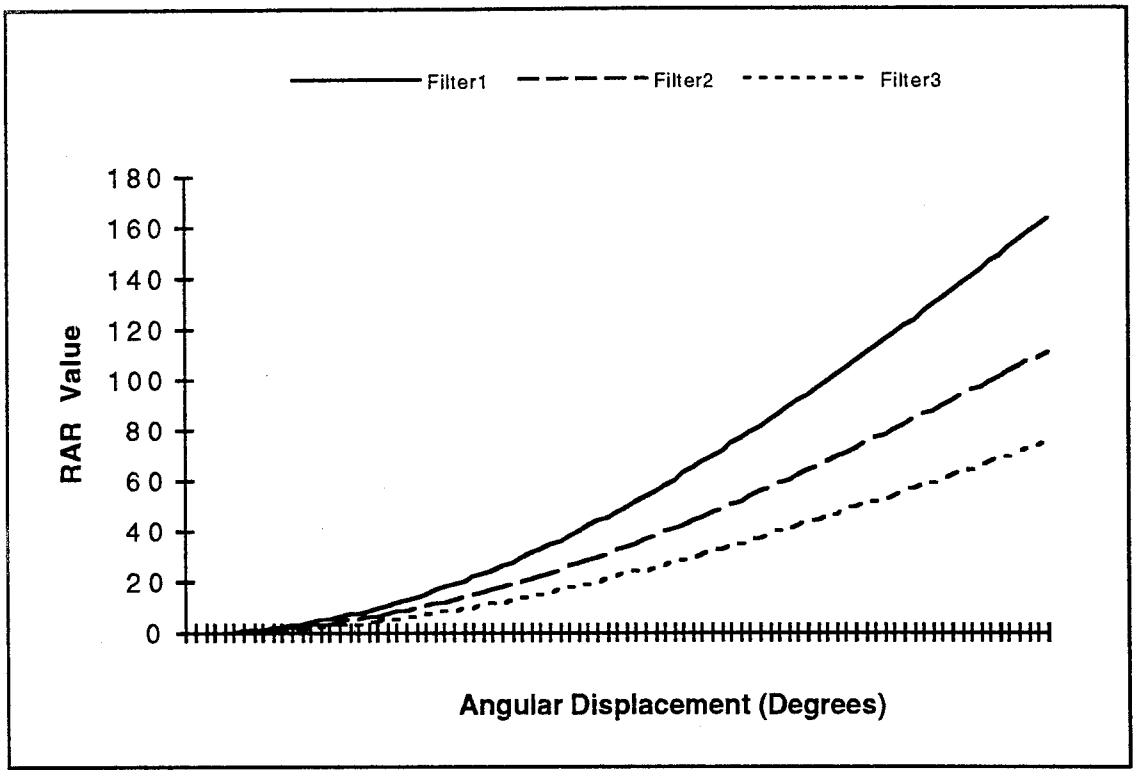


Figure 16: RAR Values for Angular Displacements through Ninety Degrees

The RAR value is also used to determine when a filter is considered divergent (when its predictions have become so unreliable that a filter restart is necessary). Based on Figure 16 and several iterations of trial and error, each of the filters was assigned a limiting RAR value. These values, and the equivalent angular displacement in degrees, are shown in Table 3:

Filter	RAR Limit	Angular Displacement
1 (Benign)	22.0	30
2 (Moderate)	33.0	45
3 (Heavy)	38.0	60

Table 3: RAR Limits and Associated Angular Displacements

After RAR values, the next performance characteristic to be examined was filter probability weights. The filters were tested under each of the motion types, and probability weights for each filter recorded and examined. Figures 16 through 18 show the probability weights associated with each filter under each of the motion types used in this research. The graphs provide insight into the behavior of the filters. The benign filter (filter #1) will absorb and retain as much probability as possible, as demonstrated by Figure 17 and the left side of Figure 18. The moderate filter (filter #2) will absorb probability if able, but does not retain it as filter #1. Instead, it will usually release its probability after a short time. This is demonstrated in Figure 18. The heavy filter (filter #3) will absorb probability if able, but will almost immediately release it again, as demonstrated in Figure 19. Note in Figure 19 that the moderate filter (filter #2) almost never gets any probability weight. The heavy filter absorbs it very quickly at the onset of heavy motion, and then the benign filter takes it all back with almost none going through the moderate filter. These filter behaviors are consistent with observed head

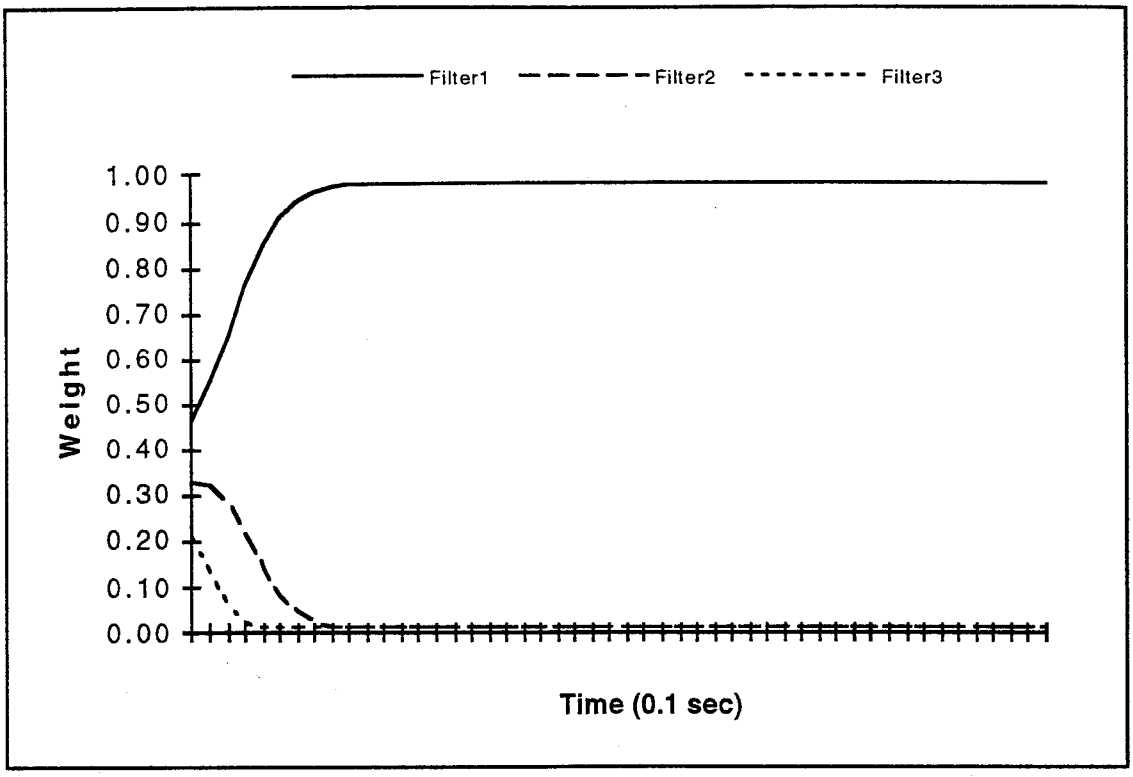


Figure 17: Probability Weights for Benign Motion Data Set

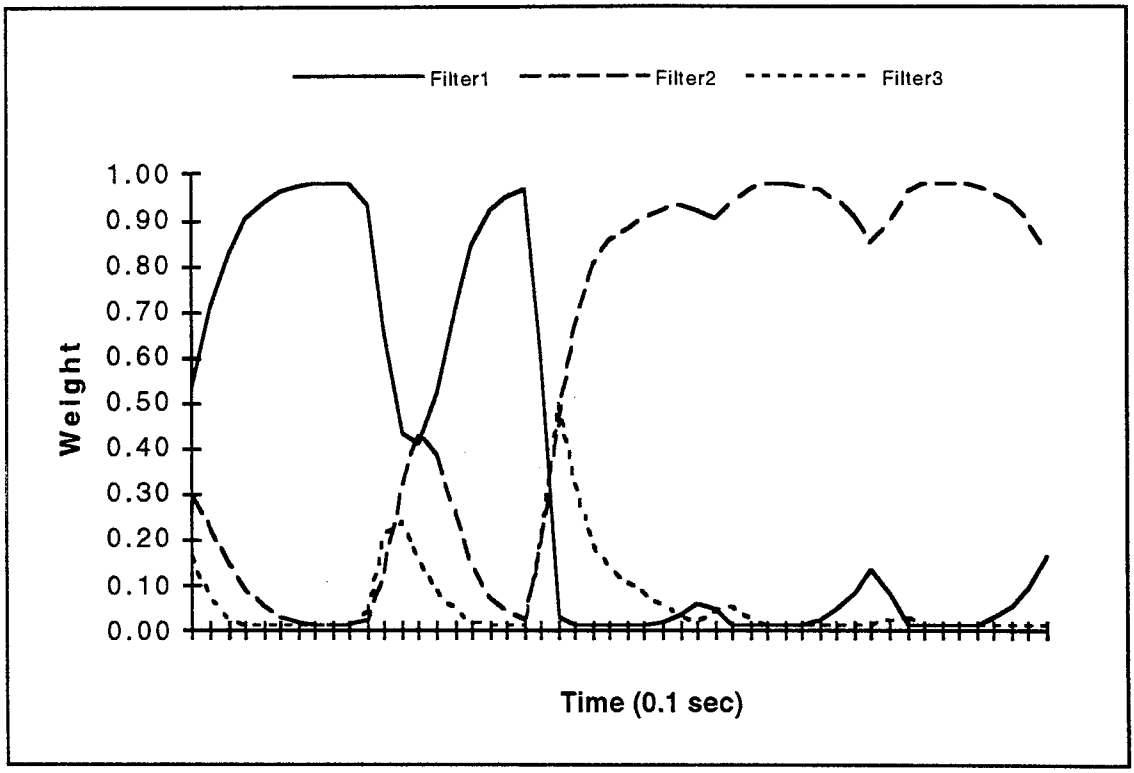


Figure 18: Probability Weights for Simulated Moderate Motion

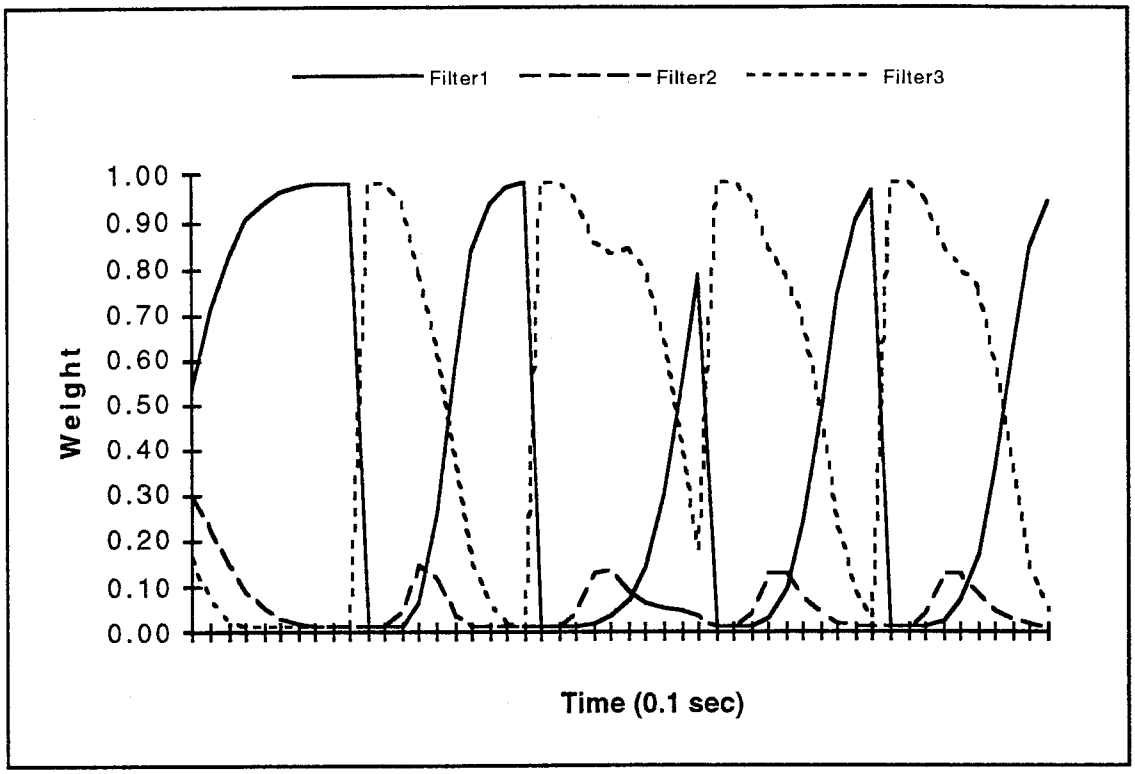


Figure 19: Probability Weights for Simulated Heavy Motion

motion patterns in humans. Head motion seems mainly composed of small (benign) movements, occasionally punctuated by sudden bursts of activity (moderate or heavy motion); however, these bursts do not normally persist for any length of time. Therefore, the benign filter (filter #1) is expected to be the dominant filter most of the time, with occasional interactions from the other filters. This characterization, however, does not preclude the moderate or heavy filters from absorbing and retaining all the available probability if the situation (as determined by the current head movement characteristics) warrants. As can be seen in Figure 18, in cases of prolonged moderate motion, the moderate filter will eventually absorb all the probability and retain it as long as the situation warrants. The same is true of the heavy filter; but these are not considered the normal cases.

The next performance characteristic examined was a measurement of the MMAE prediction error expressed as the distance between the actual data (represented as a point in 3-Space), and the MMAE prediction (represented the same way) for the same time. Figures 20 through 22 show prediction error for the corrected (zero sample periods ahead), predicted (one sample period ahead), and extended (two sample periods ahead) MMAE outputs. Note that the Predicted and Extended outputs have been offset so that the time index is correct for all three lines. Prediction error values are as one would intuitively expect. The corrected (zero sample period ahead) line shows the least error, with predictions at one and two sample periods ahead becoming increasingly error prone. However, because of the adaptive nature of the MMAE, prediction error diminishes over time. This is also

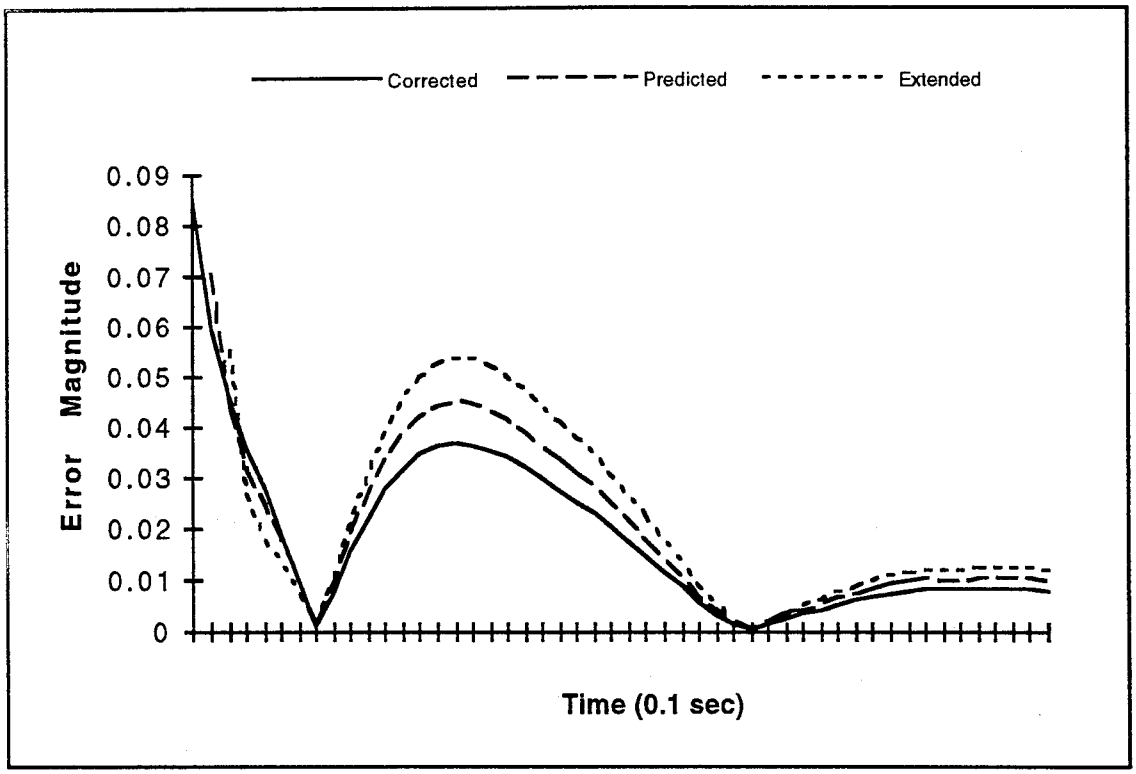


Figure 20: MMAE Prediction Error for Benign Motion Data Set

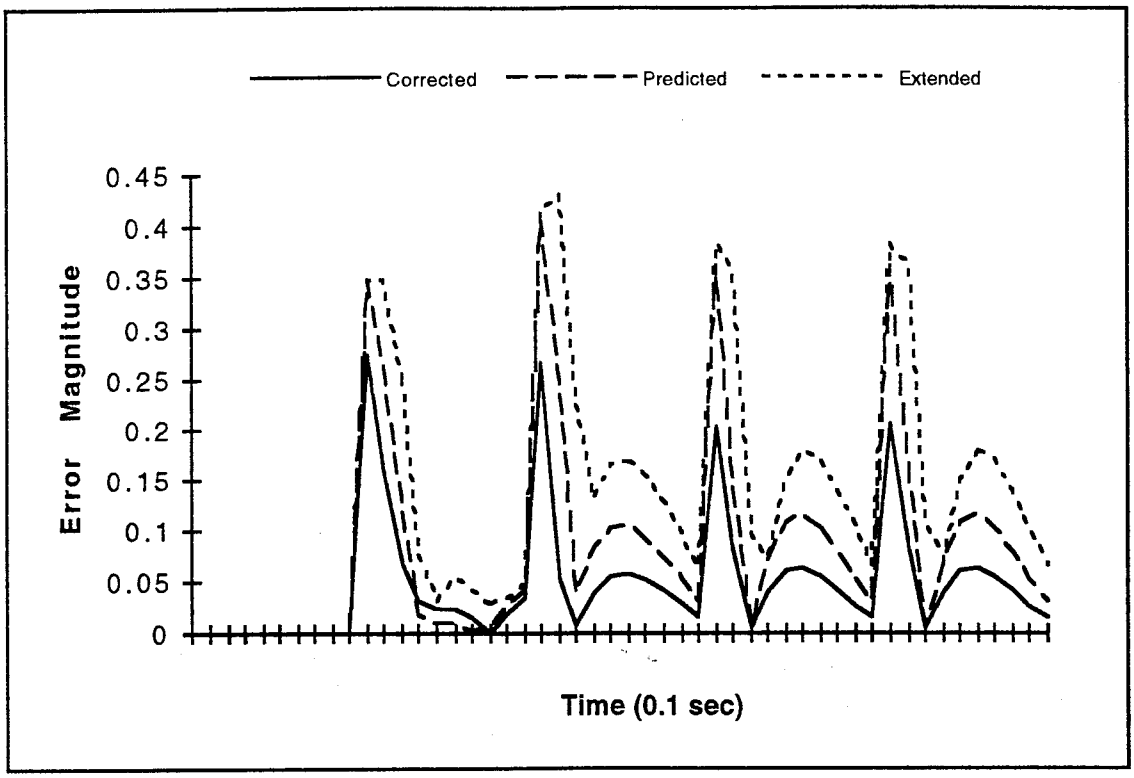


Figure 21: MMAE Prediction Error for Simulated Moderate Motion

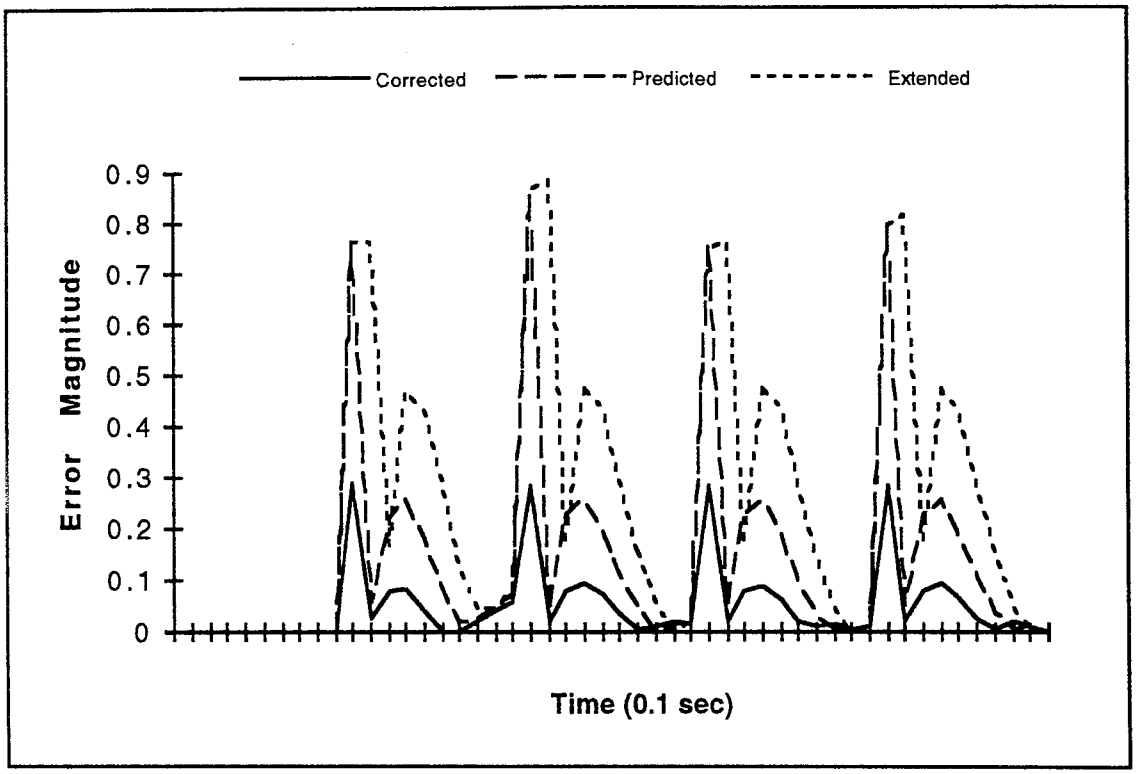


Figure 22: MMAE Prediction Error for Simulated Heavy Motion

intuitive; the longer the participant displays consistent motion, the easier it is for the MMAE to predict that motion. Note that Figure 20 was generated with a benign data set, while Figures 21 and 22 were generated through the square_wave program. This is why Figures 21 and 22 are cyclic. The square_wave program, however, is preferred since the results are repeatable.

The last performance graph compares the X-direction values (only) for an input square wave (again generated with the square_wave program) to the corrected, predicted, and extended MMAE output values. In the graph, Path is the path traced by the X-direction values from the square wave. The graph is shown as Figure 23.

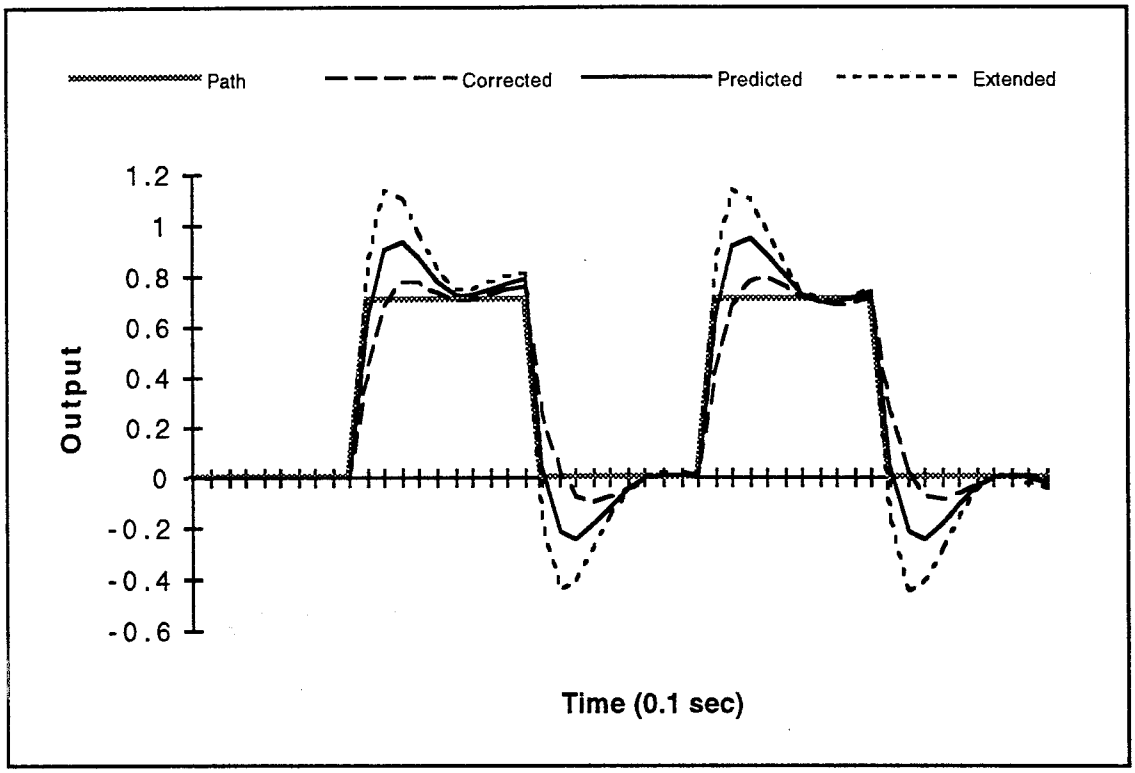


Figure 23: Comparison of X-direction Output Values

4.2 Performance Study

Once the MMAE performance was baselined, a performance study was conducted to provide a comparison of performance with actual subjects. The study was a *within subjects* study that used AFIT students and faculty for subjects. Eighteen subjects (sixteen male and two female) participated in the study. The task used for the study was an acquisition/tracking task in which subjects were asked to keep a crosshair centered on a moving 3D sphere in the virtual environment. This task was unofficially dubbed "follow the bouncing ball" by the author.

The independent variable manipulated in the study was tracking mode. Three different tracking modes (adaptive mode in which the MMAE was used to predict the orientation of the subject's head; single-filter mode in which a single Kalman filter was used to do the orientation prediction; and Polhemus-only mode in which no prediction was done) were used by each subject to perform the same task. Each subject was given seven thirty-second trials for each tracking mode; two benign trials, two moderate trials, and three mixed trials.

In the benign trial, the subject was asked to track the motion of a red ball that was constrained to stay within the subject's initial field of view (approximately forty-five degrees horizontal and thirty-five degrees vertical). The ball motion was very slow and generally followed long lines and arcs. This motion was designed to accentuate the performance of filter #1 (benign). The moderate trial was essentially the same as the benign trial with the exception that the ball motion was different. In the moderate trial, the ball still stayed generally within the subject's initial field of view, but made very short, quick movements.

In the mixed trial, two balls (one red and the other blue) were used. The red ball was positioned on the subject's left while the blue ball was on the subject's right in the virtual environment. Both balls were constrained to stay within separate areas of the virtual environment; neither ball was visible to the subject when the trial started. In the mixed trial, the crosshair had a box around it that would be either red or blue. Subjects were asked to find the ball of the same color as the box as quickly as possible and then to track that ball. The color of the box was changed either once or twice during the trial, and the box itself did not become visible until the trial started. The mixed trial was used to simulate a re-acquisition task, and thus to generate filter #3 (heavy) motion, among periods of tracking that would require use of filters #1 (benign) and #2 (moderate).

A Latin Square method with respect to the tracking mode was used to remove learning bias on the part of the subjects. The square defines the order in which the tracking modes were presented to the subjects. The square used is shown in Table 4:

Condition	Order of Tracking Modes Presented		
1	MMAE	Single Filter	Polhemus Only
2	Single Filter	Polhemus Only	MMAE
3	Polhemus Only	MMAE	Single Filter
4	MMAE	Polhemus Only	Single Filter
5	Single Filter	MMAE	Polhemus Only
6	Polhemus Only	Single Filter	MMAE

Table 4: Latin Square for Tracking Mode

Three subjects per condition were tested. Twenty-one data sets were generated so that no subject used a particular data set more than once during

testing in order to further reduce the possibility of learning effects. For Single Filter tracking mode, filter #2 was used for head orientation prediction. This is the filter originally designed for moderate head motion; it was felt that this filter would provide the best performance across all motion types.

Ten data items were collected during the performance study. Data item values were collected each frame, and written to a data file in ASCII format after the trial was completed. The data items collected are listed below.

- Ball Color – The color of the ball to be tracked.
- Red Path – The 3-space position of the red ball.
- Blue Path – The 3-space position of the blue ball.
- Z – The Polhemus input.
- Corrected X – The MMAE corrected output.
- Predicted X – The MMAE predicted output.
- Extended X – The MMAE extended output.
- RAR Value – The likelihood quotient (RAR) value for each of the MMAE filters.
- RR Value – The ME/I likelihood quotient (RR) value for each of the MMAE filters. This value is similar to the RAR value.
- Probability Weights – The probability weights for each MMAE filter.

Ball Color is a single integer (1 for red; 2 for blue); the other data items all require three output values (X-, Y-, and Z-direction values; or outputs for each of the three filters). Since the trials were thirty seconds in length, and

the frame rate of the display was set at 10 Hz, three hundred samples per trial were collected.

Two methods of analysis were then applied to the data. The first method examined the prediction error of the MMAE and Single Filter tracking modes. Prediction error looks at the absolute difference between the (MMAE or Single Filter) predicted values for a time period, and the actual Polhemus values returned at that time period. Three data sets were examined for each of the three tracking modes. In order to do comparisons for the Polhemus, a *simulated prediction error* was defined. Essentially, under normal operation, the Polhemus data used to generate a display frame is one frame old when the frame is displayed (this is, in fact, one of the contributors to overall display lag). Therefore, simulated prediction error for the Polhemus was defined as the difference between consecutive Polhemus directional values. Simulated Polhemus prediction error was compared to the actual errors in the MMAE and single Kalman filter predictions for one sample period ahead. A plot of the error in the MMAE for the Benign1 data set is shown in Figure 24. The complete set of plots is included as Appendix D to this document.

Error analysis did not use the actual ball position as a basis of comparison even though ball position is considered the "true" position data for a given sample period. This is because this study was not designed to collect information concerning error with respect to the actual ball position, but rather with respect to the Polhemus-indicated position. As an illustration of this point, consider Figure 25, which shows a plot of the actual path of the ball used in the performance study versus the Polhemus and MMAE outputs.

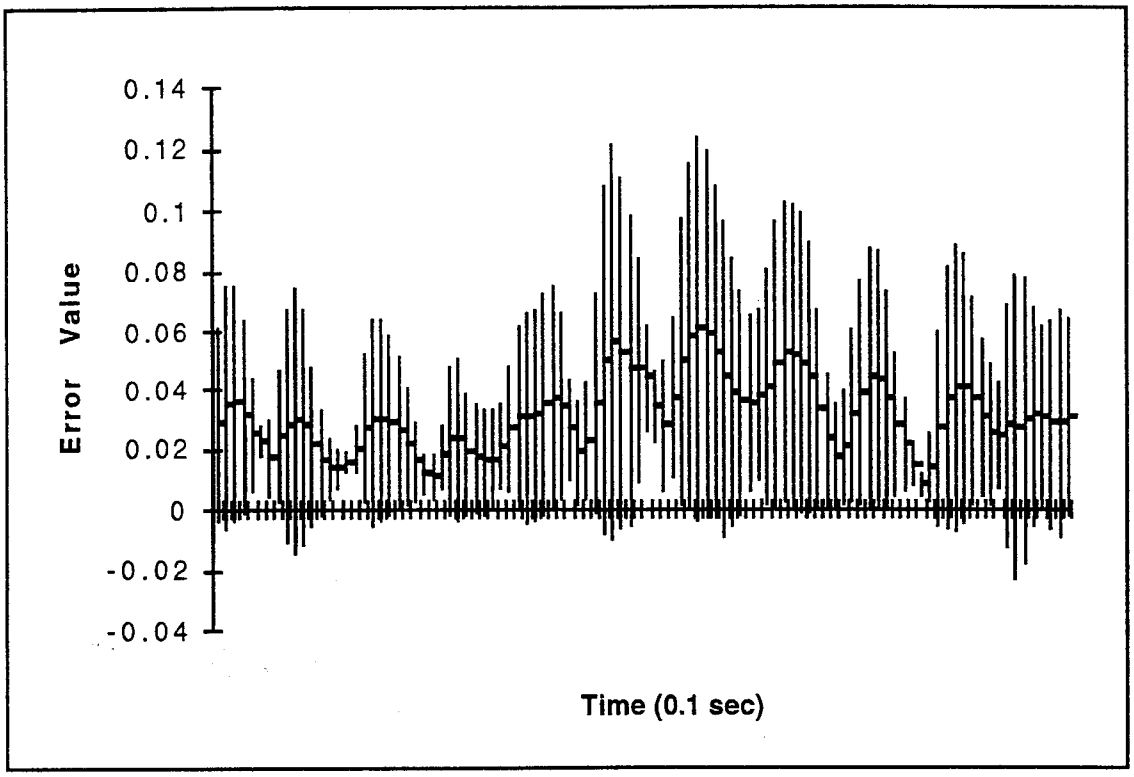


Figure 24: MMAE Prediction Error, X-direction, Benign1 Data Set

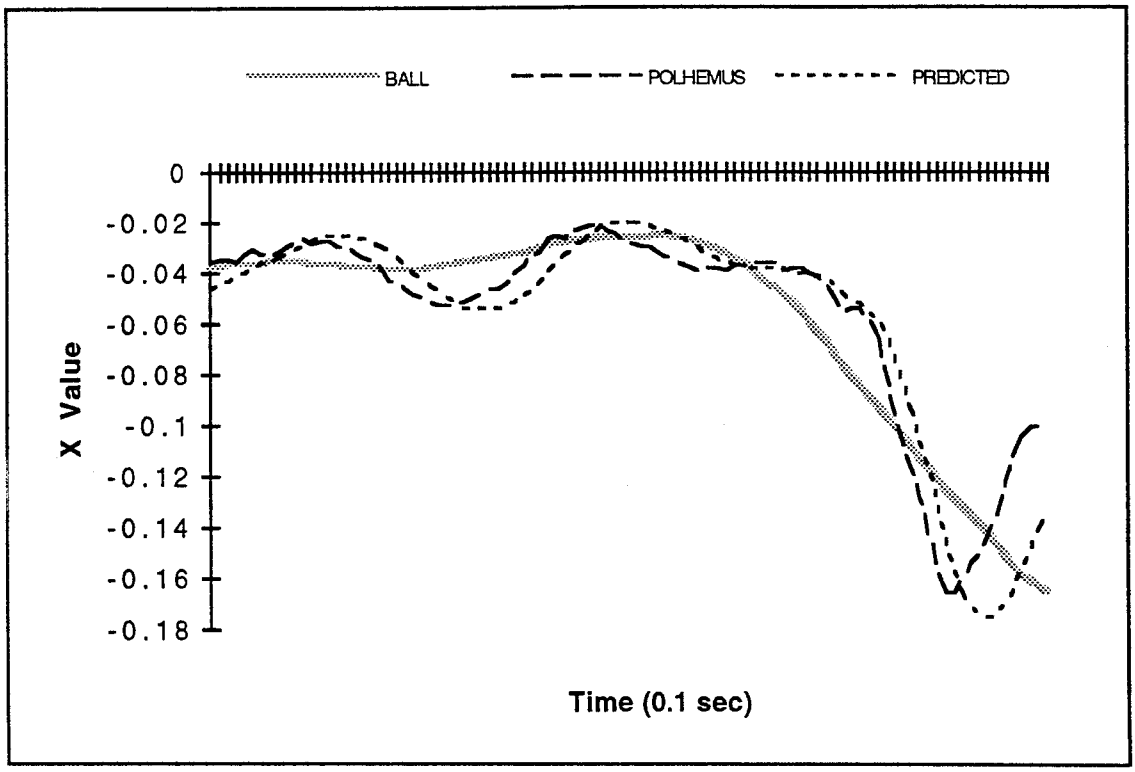


Figure 25: X-direction Comparison of Ball, Polhemus, and Predicted Values

Notice that the MMAE (prediction) line has a close correspondence to the Polhemus line. Correspondence between either the Polhemus or the MMAE lines and the actual path of the ball is much harder to identify, but it is fairly certain that a correspondence exists. It may be of some value to examine this correspondence, and see if there is a statistically significant difference between the correspondence of the Polhemus and the MMAE (and possibly the Single Filter as well) to the actual ball path.

It was expected that the performance study would produce the following results. Under benign motion, all three tracking modes (MMAE, single Kalman filter, and Polhemus Only) would appear about equal, with Polhemus Only being slightly better. Head motion in this condition is commonly so slow and predictable that the Polhemus alone is sufficient to provide a realistic display. Therefore, the MMAE was not expected to provide any advantage. Under moderate motion, it was expected that the MMAE would offer some advantage over the other two tracking modes. The Polhemus would not be able to keep up with the movement of the participant's head as well as the MMAE due to the adaptive nature of the latter. This same logic lead to the expectation that the MMAE would outperform the Polhemus in mixed motion during the reacquisitions.

Analysis of the error graphs seemed to show some trends across the different motion types, but these trends were not always what was expected. Under both benign and moderate motion, the Polhemus seemed to exhibit the smallest error values. Single Filter mode formed a middle ground, and MMAE mode seemed to have the largest errors. Under mixed motion, however, the MMAE did seem to outperform both the other tracking modes. The errors associated with the MMAE outputs show a much faster

convergence toward zero immediately after a reacquisition task (see plots in Appendix D).

A statistical analysis was done to determine if the trends noted in the prediction error analysis could be demonstrated statistically. The measure used for this analysis was sample mean and standard deviation of prediction error across all subjects for the same motion type (benign, moderate, mixed) and filter type (MMAE, Single Filter, or Polhemus Only). The formulae for sample mean ($\overline{X}$) and sample standard deviation (S) are shown below.

$$\overline{X} = \sum_{i=1}^{n} \left(x_i / n \right)$$

$$S = \left[\frac{\sum_{i=1}^{n} \left(\overline{X} - x_i \right)^2}{n-1} \right]^{1/2}$$

where n is the number of samples used. Sample means and standard deviations for prediction error of the MMAE, the single Kalman filter, and the Polhemus (this was again a simulated prediction error) were generated for each data set across all subjects. The results were partitioned into a table by tracking type and filter type. Finally, the values within each table block were again averaged to produce the results shown in Tables 5 and 6. In the tables, MMAE is MMAE tracking mode; SF is single Kalman filter tracking mode; and POL is Polhemus Only tracking mode.

	MMAE	SF	POL
BENIGN	0.0375	0.0210	0.0074
MODERATE	0.0979	0.0807	0.0440
MIXED	0.0726	0.0584	0.0286

Table 5: Sample Mean ($\overline{X}$) Values

	MMAE	SF	POL
BENIGN	0.0295	0.0166	0.0081
MODERATE	0.0595	0.0519	0.0348
MIXED	0.0592	0.0577	0.0387

Table 6: Sample Standard Deviation (S) Values

Note that in order to consolidate the data in this fashion, it was necessary to average across all samples. The results are therefore temporal values (across the entire data sets).

These results are easier to see in graphical format, so two graphs were prepared from Table 5. The first (Figure 26) treats tracking type as the parameter. Although it appears that there is a significant difference between the lines portrayed, in fact the sample standard deviations are so large that all the lines are statistically equivalent (based on visual inspection). The same conclusion is reached for the second plot (Figure 27), which treats motion type as the parameter. Again based on visual inspection, no statistically significant difference between the tracking types is noted.

The overall conclusion to be drawn from the analyses is that the performance study did not produce any statistically significant results; at least, not at the level of analysis done. There are many possible reasons for this. One of the most probable is that the study, as performed, lacked the experimental power to show any differences between the conditions. The tracking task used in the study is extremely difficult. Referring again to Figure 25, the low correlation between the path of the Polhemus (and the MMAE) and the path of the ball indicates that, overall, the subjects had a great deal of difficulty with the task. This in turn could mean that the number of subjects and trials used was insufficient to identify any

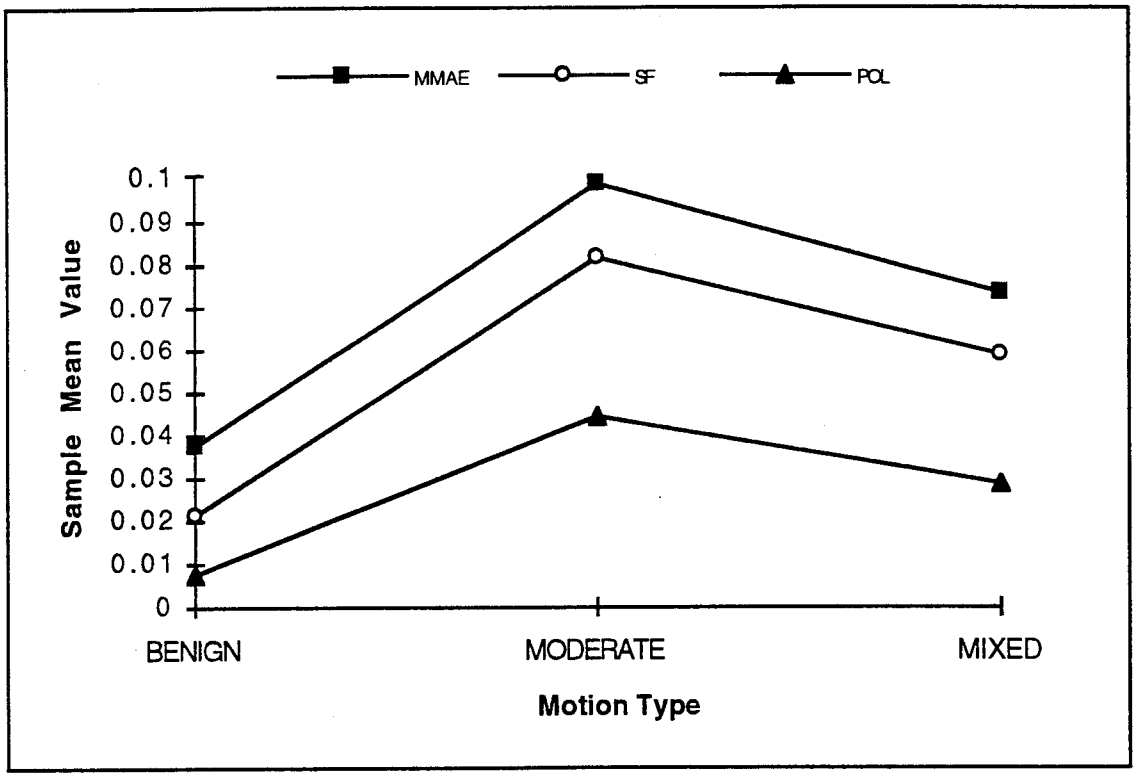


Figure 26: Plot of Sample Means with Tracking Type as the Parameter

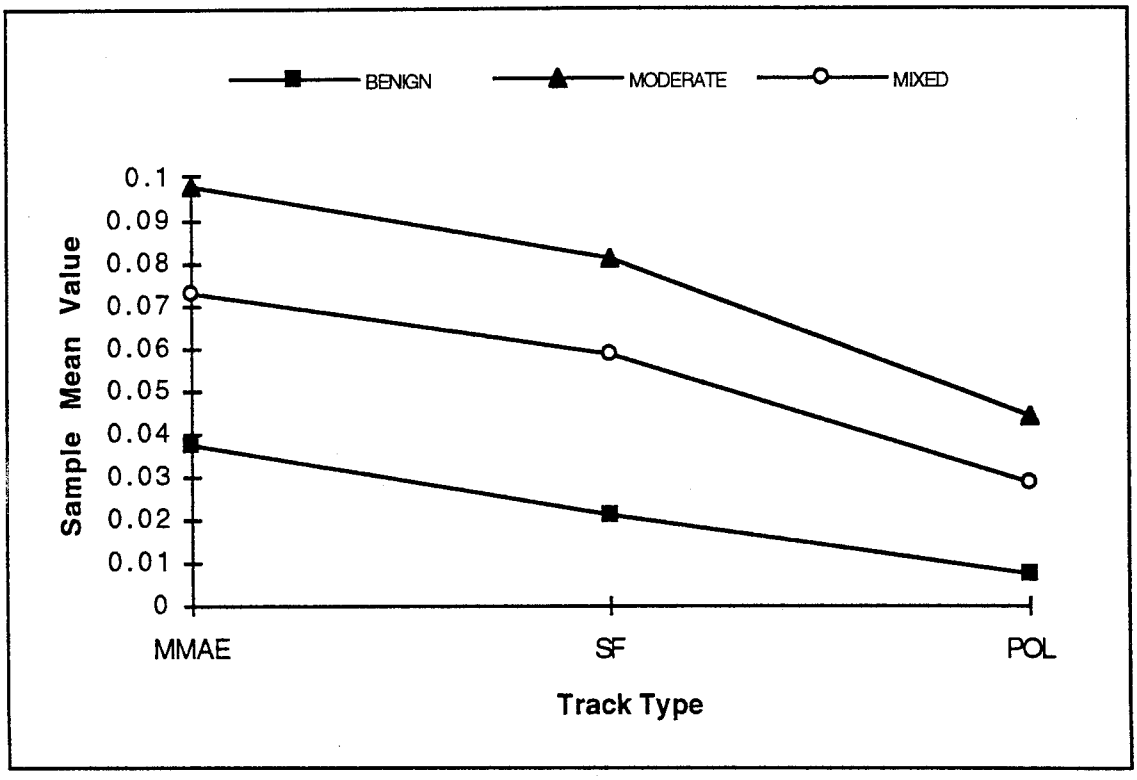


Figure 27: Plot of Sample Means with Motion Type as the Parameter

performance differences. Another possible cause is using the Polhemus data as a basis for comparison. This data, although it was treated as "true" head location, is actually subject to the considerable noise in the environment of the AFIT graphics laboratory. A final possibility is that there is some information to be gained from the data, but that more detailed analysis of the individual data sets will be necessary to find it.

4.3 Language/Performance Comparison

One of the goals of this research was to compare an Ada 9X implementation of this software to a C++ implementation. To meet this goal, two versions of the software (Ada 9X and C++) were developed. The two versions were then compared both by the methods described in the previous section, and also by comparing operational (performance) characteristics and design characteristics.

Implementation Comparison. Since the application was developed first in C++, the Ada 9X implementation was much easier. Instead of having to examine test outputs for correctness, they could be directly compared to the already existing outputs from the C++ implementation. This provided both a verification of the Ada 9X implementation, and an extra validation of the C++ implementation.

As with the C++ implementation, sample outputs from test runs of the Ada 9X implementation are included in Appendix B of this document. Note that the outputs are not *exactly* identical across the implementations. While they are largely identical, there is some variability in the filter outputs. This was attributed to differences in floating point number representation and/or differences in library mathematics functions such as square root or exponent.

Performance Comparison. Recently, graphics intensive software applications have used C and C++ as the implementation language. This has led to the gradual development of a large software base for graphics applications written in C, which has in turn produced a great deal of inertia toward the continued use of C++ as an application language. While this is understandable, many of the factors that drove that language decision have changed in recent years, and it may be time to re-examine those decisions in light of those changes.

Ada 9X, although still relatively immature, has much to offer to virtual environment applications. Two of the more immediately obvious benefits are strong typing and exception handling. These two combine to make Ada 9X a highly reliable language. The developer can constrain the ranges of variables to the exact real-world values needed for the application, and range constraints are checked automatically by the run-time environment. This allows the developer a high level of confidence that the application variable contain valid values. Exception handling has two important benefits. First, it separates normal processing from exceptional, or abnormal, processing. This makes the code more readable, since exceptions are not interspersed with normal operations. Secondly, it allows a called routine to raise an exception to the calling routine, thus placing responsibility for dealing with the exception where it belongs—at the higher (and potentially highest) level routine in the application.

There are essentially two common arguments against using Ada as an implementation language. The first of these is that Ada is not object-oriented, so cannot be used for object-oriented programming. While this was

true for Ada 83, the revised Ada 9X is a fully object-oriented language, and can provide the support necessary for object-oriented development.

The second argument is that applications developed in Ada simply run too slowly; the performance degradation is too great to make the language a viable choice. One of the goals of this research was to provide a comparison of equivalent, computationally intensive applications to see what that performance degradation actually is, if one exists at all.

To conduct the comparison, the Ada and C++ versions of the application were compiled and executed on an SGI Reality Engine^2 platform running under the IRIX 5.2 operating system. The data collected from the test runs is summarized in Table 7.

	C++ IRIX4	C++ IRIX 5.2	Ada 9X IRIX5.2
3SPACE read time (msec)	38	17	17
3SPACE read reliability (percent)	100%	100%	100%
MMAE cycle time (msec)	27	9-23**	20
Filter cycle time (msec)	8	4	7
Frame Rate (Hz)	20	30	30
Executable Size (bytes)	N/A	413,356	819,056

** This value was observed to have large variability in the measurement values obtained, so the range is reported here

Table 7: C++/Ada 9X Performance Comparison Data

Data gathered from test runs of the thesis software under IRIX 4 are included in the table for completeness.

Polhemus read times and reliability measures were determined with the TS_time (ThreeSpace time) program, which performs 1000 successful Polhemus reads. Since it is possible that a given read will not be successful (the data is not ready when the Polhemus is read), TS_time reports how

many read attempts had to be made in order to achieve 1000 successful reads, and also how long it took (in seconds) to accomplish those reads. A sample size of 1000 was used so that the output of the program could be translated directly into milliseconds (msec) per read. TS_time was run at least ten times for each of the implementations. The data shows that, under IRIX 5.2, both C++ and Ada 9X read data from the 3SPACE at approximately 60 Hz with 100% reliability. This is the maximum output rate of the Polhemus 3SPACE magnetic tracker used in this research, so no determination of speed above this maximum could be made.

The MMAE cycle time was determined with the mmae_time program, which performs 1000 iterations of the same three calls. First Correct is called with a (constant) Z value, then Predict, and finally Project is called for a projection two sample periods ahead. mmae_time reports the number of seconds required to do the cycle 1000 times. The data in Table 7 represent the mean value from ten executions of the test program. The C++ version showed a great deal of variability in the cycle time; values from a low of nine to a high of twenty-three were noted during testing. Therefore, the range is reported in the table.

Filter cycle time was determined with the filter_time program, which is conceptually similar to mmae_time, above. 1000 iterations of the filter routines (Update, Propagate, and Project) are performed, and the time required in seconds is reported. Again, the data in Table 7 represent the mean of ten program executions. Here, there was very little variability noted in the figures obtained. Results indicate that the C++ implementation executed slightly faster than the Ada 9X implementation.

Frame rate was determined by modifying the pfdisplay application to use Free-Running mode and a desired frame rate of 60 Hz. In Free-Running mode, frames are rendered as quickly as they are produced (therefore frame rate is maximized). The precise choice of frame rate is not that critical; Performer will use whatever frame rate it can. However, 60 Hz is the typical goal frame rate for graphics applications, so it was used. The actual frame rate for the application was noted by having Performer display channel statistics during program execution. These statistics give real-time indications of application performance in terms of frame rate and time spent in the various processes (application time, render time, etc.). In all trials, the window size used for the display was equal to the screen size available (1280x1024 pixels). Also, the applications were verified to have an application time (computation time) greater than the draw time (display render time), thus eliminating the possibility that the frame was driven solely by how long it took to render the scene. The values reported in Table 7 represent modal values (the values most commonly seen), since some frame rate variability was noted.

The final row, executable size, was taken from a directory listing of the executables. The C++ executable is approximately one half the size in bytes of the Ada 9X executable. This could be due to the immaturity of the Ada 9X compiler used.

The table is somewhat surprising. It was expected that the Ada 9X implementation would not be able to generate performance equal to the C++ implementation; the Ada 9X compiler is immature, and the Ada 9X software had extra layers of interface from the Performer bindings. However, the frame rates for both applications are identical (with some variability in the frame rate; however, this variability was noted in both applications). Also,

current graphics applications in the AFIT graphics laboratory run at approximately 12 to 15 Hz, which can easily be met by either implementation. The overall conclusion is that, based only on this data, Ada 9X implementations appear to have the same performance capabilities (in terms of frame rate) as C++ implementations.

The other comparison performed was a design-based comparison. The metric used for this comparison was *Weighted Methods per Class* (WMC) developed by Chidamber and Kemerer [Chidamber91]. This metric measures the complexity of a class by the sum of the static complexities of the class members. The formula for this is given below.

$$WMC = \sum_{i=1}^{n} C_i$$

where C_i is the static complexity of each method within the class. Chidamber and Kemerer did not offer a preferred method for determining static complexity, so McCabe's Complexity Metric [McCabe89] was used. The results are summarized in Table 8.

	C++		Ada 9X	
Class	#Methods	WMC	#Methods	WMC
GenMatrix	22	72	13	58
FOGMA_Filter	18	44	16	49
FOGMA_MMAE	18	64	15	61
ThreeSpace	3	7	3	5
Totals	61	187	47	173

Table 8: Results of WMC Calculations for Implementation Classes

Note that since the abstract classes (MMAE, Kalman_Filter) do not have implementations (only specifications), they are not included in the table.

McCabe states that the complexity of a unit should be kept under ten [McCabe89]; complexities greater than ten usually indicate software that needs to be redesigned to reduce its complexity. Both of these implementations are well below this threshold; the overall average complexity is 3.1 for the C++ implementation and 3.7 for the Ada 9X implementation.

Overall, the table indicates that the two implementations are very similar. This was the expected result, because the Ada 9X was purposely implemented to be as structurally similar to the C++ as possible. McCabe's Complexity Metric [McCabe89] is essentially a structurally-based complexity measure, so the similarly structured implementations should have similar complexities as measured by McCabe's. It is likely that re-implementing the Ada 9X design to take better advantage of the software engineering principles embodied in the language would significantly reduce the complexity measure for the Ada 9X implementation. As an illustration of this point, consider the following. The Ada 9X implementation did not use constrained ranges for variables since these are not available in C++. If the constraints were used, then many bounds checks could be eliminated from the Ada 9X implementation (they would be handled by the run-time environment automatically). Since McCabe's complexity measure relates directly to decision points in the methods (such as deciding whether or not a value is within the prescribed range), moving these bounds checks out of the method would have a direct impact on the complexity measure.

4.4 Summary

This chapter presented results and analysis of the studies conducted to validate this research. Essentially, the studies used fall into three broad categories: studies done during implementation of the software; studies done to provide a characterization of the MMAE performance in comparison to other tracking modes (Single Filter, Polhemus Only); and studies done to provide a comparison of the implementation languages (C++ and Ada 9X).

Studies done during implementation had three main objectives. The first objective was to verify proper functioning of the software. To this end, several test programs were developed that allowed the system objects to be individually tested and studied. The second objective was to tune the individual filters. This was accomplished through a trial and error process in which a researcher would generate an appropriate motion type with the Polhemus and then observe the response of the MMAE. If the MMAE response did not meet expectations, then adjustments to the individual filter σ and τ values were made. This process was continued until a satisfactory MMAE tuning was achieved. The third and final objective was to establish a performance baseline for the MMAE. This was achieved through the use of the previously mentioned test programs and prepared data sets for the various types of motion used in the research. The data from these tests was used to establish a performance baseline for future researchers.

In order to characterize the performance of the MMAE in comparison to other tracking modes, a performance study involving AFIT faculty and students as subjects was conducted. The task used during the study was an acquisition/tracking task in which subjects were asked to follow the motion of a 3D sphere in a virtual environment. Ten different data items were collected

during the study, and the resulting data sample analyzed for any trends. The raw data suggests that the Polhemus may be the best overall choice in tracking mode; however, this observation cannot be shown statistically, and it is more likely that the study format was not sufficiently sensitive to detect differences between the tracking modes.

The implementation language comparison had two objectives. The first involved comparing the performance of two equivalent applications, one developed in C++ and the other in Ada 9X. Two implementations were done, and then compared under several criteria. Other than executable size (in which the C++ had the clear advantage) the implementations provided virtually identical performance. The discrepancy in executable size was attributed to compiler immaturity on the part of the Ada 9X compiler.

The other objective was to provide a measure of design quality. This was provided by a comparison of Weighted Methods per Class, a metric developed by Chidamber and Kemerer [Chidamber92]. This metric assigns a weight to each class in an object-oriented design according to the number and complexity of methods within the class. Method complexity was calculated according to McCabe's Complexity Metric [McCabe89]. The results showed that the two implementations have about equal complexity, and that both are well within the low complexity range as specified by McCabe [McCabe89]. This was the expected result; the Ada 9X implementation purposely tried to mimic the structure of the C++ implementation, so a complexity measure based on structure (such as McCabe's) should report the implementations roughly equal. The Ada 9X implementation could be altered to take better advantage of the software engineering principles embodied by the language,

and this could have a significant impact on the complexity measure of the resulting implementation.

V Conclusions and Recommendations

The focus of this research has been to apply software engineering tools and techniques to develop an application that reduces the lag typically present in virtual environment displays. The application developed was a Multiple Model Adaptive Estimator (MMAE) composed of three Kalman filters that predicted the orientation of a participant's head one sample period into the future. The prediction was used by the environment rendering software to generate the image shown to the participant. The period used for the MMAE prediction was equal to the time required to compute and render the next frame for the environment display. Assuming perfect prediction, this approach would allow the computer to generate and display a virtual environment with zero lag (therefore in real-time); the frame shown to the participant would be appropriate to his/her current head orientation at the time the frame was displayed.

Each of the filters in the MMAE was designed for a different type of head motion (benign, slow movements; moderate, normal movements; and rapid, jerky movements), thus allowing the MMAE to adapt to changes in head movement characteristics. Each filter in the MMAE produced a predicted orientation and a probability value that the hypothesis used in designing the filter was the best hypothesis about the participant's head motion for that prediction. The MMAE used the probability outputs to assign probability weightings to each filter's output. These weightings reflected the probability that the hypothesis used in a particular filter was correct for the observed behavior. The MMAE estimate was then computed as a probability weighted average of the individual filter outputs.

The focus of the software engineering aspect of this research was to investigate the appropriateness of Ada 9X as a software engineering tool and implementation language for a virtual environment application. Recently, C and C++ have been the implementation languages of choice for these applications. However, Ada 9X provides all the object-oriented features necessary for design and development, and also offers engineering advantages that might make it more attractive in the future. Strong typing and the ability to use exceptions to separate normal processing from exception handling are examples of the engineering support provided by Ada 9X.

The research approach was validated by two studies. The first was a comparison of C++ and Ada 9X implementations of the application software. The performance of the application in terms of maximum allowable frame rate and time required for individual components to execute was compared. Also, the extent to which the implementations support software engineering principles was also compared. The other was a performance study of the application. Subjects taken from AFIT faculty and staff were asked to follow the movements of a ball (3D sphere) in a virtual environment. Several methods of tracking the subjects' head movements were compared: Polhemus only tracking (i.e., without any prediction); single Kalman filter based tracking (i.e., a predictive filter with no adaptive capability); and MMAE tracking. The purpose of the study was to validate the approach taken in designing and implementing the MMAE , and to provide a performance baseline against which to measure future research efforts.

Several results were generated by the studies. Data collected during implementation of the software was used to establish a performance baseline

for the MMAE developed in this research. This will give future researchers in this area a benchmark against which to measure their own efforts. The performance study data indicated a possible advantage for the MMAE under the assumption of moderate motion, but this observation could not be shown to be statistically valid. More, and more explicit, research is needed in this area to provide a valid, definitive characterization of the ability of the MMAE compared to other tracking modes. Finally, the language study had the surprising result that Ada 9X appears to perform just as well as an equivalent C++ implementation under IRIX 5.2. This result is very surprising because of the immaturity of the Ada 9X compiler, and the extra interface software required to develop the Ada 9X implementation. This result, combined with Ada 9X's strong support of software engineering practices and principles, makes an argument for the use of Ada 9X as an implementation language for graphics-intensive applications.

Although much work has been done to this point, and some observations and conclusions made, there are still many discoveries waiting in this field of research. The following paragraphs present recommendations for further research in this area. They are based on experience and insight gained during the course of this research, and are divided into software engineering, Kalman filter/MMAE, and performance areas.

5.1 *Software Engineering Recommendations*

There are many opportunities for further research in the software engineering and implementation area; only a few are presented here.

- Re-examine the Ada 9X implementation in light of (anticipated) compiler improvements. The GNAT compiler used in this research is

currently very immature, and this was one of the constraints that drove the Ada 9X design. It is hoped that the compiler will become much more stable and mature in the near future, and it would then be appropriate to reexamine the Ada 9X design and implementation in light of the increased capabilities of the compiler.

- Continue to use the Ada 9X implementation as the basis for future research efforts. The operating system under which this thesis work was done is due to be upgraded, and it will be necessary to upgrade the C++ software to make it compatible with the new operating system if it is desired to use the C++ implementation for future research. By contrast, the Ada 9X implementation offers virtually identical performance, and is already fully operational under IRIX 5.2.

- Decouple the Polhemus interface from the main application. One of the known problems with Kalman filters and MMAEs is their tendency to overshoot target motion. While this can be minimized through proper tuning of the individual filters, it cannot be entirely eliminated. However, it may be possible to minimize the impact of overshoot through oversampling of the MMAE. If the MMAE is sampled and updated at a rate greater than the actual display update rate used (2 or 3 times as often, for example) then it may be possible to get through the overshoot period before the display is updated. This, at least in theory, could provide the participant with the appearance of a much better prediction model. One method of achieving this oversampling is to remove the Polhemus and MMAE objects from the main application, and make them a separate, independently executing process. The Polhemus could provide information to the MMAE as rapidly as possible, and the MMAE could place the results in a shared memory block for the renderer to take whenever it was ready.

- Enhance the display characteristics of the virtual environment. Two possible enhancements are: to include roll cues in the display, and to modify the display (and appropriate outputs) so that filter weights are shown as zero in Polhemus Only mode. Roll was not included in the head motion for this research because calculating roll required reading

extra information from the Polhemus, and the resulting increase in read time was unacceptable. However, the read time decreased dramatically when the software was executed under IRIX 5.2, and reading the extra information became a possibility. The realism of the display will be greatly enhanced if roll cues are provided. The other enhancement (outputting filter weights of zero in Polhemus Only mode) will make it easier for the researcher and the participant to differentiate between Polhemus Only and MMAE modes (currently, the filter weight outputs for these modes are identical). This would be of tremendous benefit when analyzing data from performance studies. The researcher could have a reasonable assurance that the correct MMAE mode was used simply by examining the filter weight outputs.

5.2 *Kalman filter / MMAE Recommendations*

The MMAE developed in this research was a necessary first step into the field; a stepping-off point for further research. The recommendations below outline some of the possibilities in this area.

- Refine the MMAE developed in this research. Because this is the first research in this area done at AFIT, one of the goals of this research was to provide a performance baseline against which to compare future applications. However, the performance baseline developed can just as effectively be used to improve the performance of this application. It seems reasonable to fine tune this application to the greatest extent possible in order to provide the best possible basis for further work.

- Explore other models. The First Order Gauss Markov acceleration model used in this research may not be the best dynamics model for head motion. A logical direction for future research is therefore to explore other models such as First Order Gauss Markov Velocity models. On a related note, it is possible that the Constant Gain Filter is also not the best choice. This research had the luxury of using a

fixed frame rate, which also made the filters used in this research viable. However, other graphics research applications at AFIT do not use a fixed frame rate; instead, the renderer is asked to provide the maximum frame rate possible for the scene being rendered. The result is that frame rates can vary wildly during program execution. Kalman filters that do not require a fixed time interval for doing predictions may provide more accurate and useful predictions in these situations, and should be explored.

- Compare the MMAE to other predictor models. The MMAE developed in this research is only one approach to solving the head prediction and lag problems. MIT researchers have developed an MMAE based on different designs (velocity-based dynamics models within a maximum a posteriori MMAE) that is used with their virtual drum set application [Friedman92]. It is reasonable to believe that there is much to be learned from doing a fair comparison of these approaches.

5.3 *Performance Study Recommendations*

The final area for recommendations is the area of human performance. This area was not treated in any great depth in this research, but is certain to become much more important in future research. The ultimate test of an engineering process is the product that it produces. When that product is a tool such as the MMAE developed in this research, it must eventually be measured by its ability or inability to improve the performance of a task. The product of this research is no different; it will have to be measured by its ability or inability to provide a participant with a more realistic virtual experience. In keeping with these observations, the following recommendations are made:

- Characterize MMAE performance in terms of enhancing the ability of a human to perform a task. The studies done in this research provide a good basis for analyzing and improving the performance of the MMAE as a head movement predictor; however, it is desirable to measure the benefit, if any, to be gained from the MMAE in terms of task performance in virtual environment.

- Design experiments that measure the ability of the human to perform a task, not just the ability of the MMAE to predict head motion. An example of such a task is to have a ball appear at a random location in the virtual environment, and then begin moving in some (possibly pseudo-random) manner. In order to successfully complete the trial, the subject must find the ball as quickly as possible, and then successfully track it for a set amount of time (one second for example). Collect data on how long it takes the subject to successfully complete the trial. Use different tracking modes in the trials. These types of tasks allow the benefit of using the MMAE versus some other form of tracking/prediction strategy to be measured and analyzed.

- In a related note to the one above, use more real-world tasks to perform the human performance studies. Research is always best when it is applied to real-world problems with real-world constraints. Select experiments that not only give good indications of the MMAE performance, but that also closely mimic real-world applications of this research.

5.4 The Final Word

The final recommendation does not fall into any of the categories defined above. This line of research involves several different areas of study and expertise, and the potential for fruitful research in each of these areas is enormous. However, the investment in time and energy to gain sufficient background in each of these areas is also enormous. In order to minimize the

time necessary to gain sufficient background knowledge to do this research, it seems reasonable to work in teams rather than as individuals. This will allow the teams to increase their productivity by using the experience of the team members.

Initial research efforts can be done in teams of two (one electrical engineering student to handle the technical end of the MMAE design and implementation, and one software engineering student to handle the software design and implementation). Each of the students can write an individual thesis document to meet AFIT graduation requirements. Later, if the line of research proves as fruitful as it promises to be, and if other issues (such as performance issues) are ready to be explored, other students can be added to the team. Again, the potential of this line of research is enormous; certainly big enough to allow exploration in teams.

Appendix A:
Prototype Software Problem Reports

The following is a listing of various problems encountered in the prototype software, and the actions taken (if any) to correct them. This listing is provided for two reasons. First and foremost, these problem reports provide visibility to the factors that drove design decisions for the thesis software. The insight and experience gained from modifying and using the prototype software was instrumental in determining how to design and implement the thesis software. Second, these reports serve to underscore the importance of thorough software design and testing strategies. It is reasonable to assume that the original implementers of this software did not intend these problems to be present in the final product; therefore, we can conclude that whatever design and testing strategy they employed did not uncover these problems.

Each of the entries below contains the date the problem was first reported, the person or persons who discovered it, a general description of the problem, and a list of actions taken to isolate and correct the problem. The problem reports are listed in chronological order by discovery date.

DATE 1 January 1994
FINDER Russell
PROBLEM The filters in the MMAE produce nonsense values; residual values are HUGE.
ACTIONS

- 15 February 1994. Changed **cnt** value (minimum number of filter cycles before restart may occur) from 15 to 10 to allow faster restart of divergent filters. No noticeable effect on performance.
- 22 February 1994. Removed lower bound check on exponentiation in calculation of f-value. No noticeable effect on performance.
- 24 February 1994. Added code to zero **PMAT1**, **PMAT2**, and **PMAT3** before initialization. Necessary because 1) C++ does not clear memory

at allocation, and 2) not all matrix elements are given a value at initialization. This corrected the residual and divergence problems.

- 25 February 1994. Re-tuned filters.
- Addendum. The prototype code did not have a separate matrix object; instead, matrix operations were defined inside the code for the Kalman filter.

DATE 1 January 1994
FINDER Russell
PROBLEM The probabilities for the individual filters sum to more than 1.0.
ACTIONS

- 22 February 1994. Rewrote code which enforces lower probability bound of 0.01. Original code did not correctly handle the case where two of the three filters needed to be adjusted.

DATE 28 February 1994
FINDER Amburn
PROBLEM The box in the center of the display needs to be changed to crosshairs.
ACTIONS

- 28 February 1994. Changed box to crosshairs.

DATE 31 March 1994
FINDER Maybeck
PROBLEM The renderer is not sensitive enough to accommodate small changes in head orientation, resulting in coarse movement.
ACTIONS

- 31 March 1994. Removed the **EPSILON** value check in the renderer (the view direction into the scene was not changed until the orientation values changed by at least **EPSILON**). This resulted in acceptable movement characteristics.

DATE 31 March 1994
FINDER Amburn, Maybeck

PROBLEM The aircraft in the environment appear to undergo shearing when they make clearing turns. They become very difficult to follow when this occurs.

ACTIONS

- 31 March 1994. Believe the problem is in the orientation vectors for the aircraft contained in the data file. Investigating methods to guarantee orthogonal orientation vectors.
- Addendum. The problem was indeed in the data files. No fix to this problem was ever implemented.

DATE 31 March 1994

FINDER Russell

PROBLEM The renderer inverts up and down when the viewer is at or beyond ninety degrees from initial forward direction.

ACTIONS

- 31 March 1994. Investigated Polhemus orientation to assure we are in the active hemisphere of the Polhemus.
- 30 April 1994. Rewrote read_Adjusted_Fastrak method to return readings in row-major ordering (instead of column-major); verified right-hand coordinate system. This resulted in acceptable performance.
- Addendum. read_Adjusted_Fastrak is a method in a library object for interfacing with a Polhemus. In order to make this fix, it was necessary to first copy the library code to a local directory, then modify the local copy.

DATE 31 March 1994

FINDER Maybeck

PROBLEM The status information in the display undergoes a color intensity shift (white to gray to black) depending on the color of the terrain behind it.

ACTIONS

- 4 March 1994. Changed information display color to black. This eliminates the intensity shift (black has no intensity). Cause of original intensity shift is still unknown.

- Addendum. The cause for the original intensity shift was never isolated.

DATE 31 March 1994
FINDER Maybeck
PROBLEM We need to be able to isolate the filters so that only one of them is running (single filter mode). This will aid in filter tuning.
ACTIONS

- 8 April 1994. Added **MMAE_Mode** variable to allow for three modes of operation: polhemus mode, single filter mode, and adaptive filter mode. Added 'm' option to checkInput routine to allow the mode of operation to change dynamically.

- 6 May 1994. Added dummy probabilities to MultModels method. These probabilities allow the MMAE to simulate single filter mode without incurring the performance degradation associated with actually shutting down filters.

- Addendum. The prototype code did not have separate Kalman filter and MMAE objects; instead, a single Kalman filter object was coded the encompassed the behavior of both. This single object had duplicate variables so that it could simulate three Kalman filter objects, as well as code to handle necessary MMAE functions. Because of this, it was very difficult to isolate problems with and/or make modifications to either the filters or the MMAE.

DATE 31 March 1994
FINDER Maybeck
PROBLEM The data currently collected by the simulation is not sufficient.

ACTIONS

- 22 April 1994. Added individual filter outputs, **sim_time**, **big**, and **little** to the output data. Made necessary adjustments to routines.
- 26 April 1994. Added **VRP** and **VPN** to output data.
- 28 April 1994. Added **cycle_start** and **cycle_end** to output data.
- Addendum. Adding these output variables involved changes to approximately six routines in several different objects, including the use of shared memory. The effort required to make these changes is directly related to the coupling between objects in the prototype code.

DATE 20 April 1994

FINDER Russell, Maybeck

PROBLEM In follow mode, the participant's distance behind the aircraft is not fixed, but is a function of the aircraft's velocity.

ACTIONS

- Addendum. No actions were taken on this problem.

Appendix B:
Test Program Sample Outputs

This appendix contains output listings for the filter_test and mmae_test programs in both implementation languages (C++ and Ada 9X). The benign filter was used for both of the filter_test executions, as was the Benign0 data set. The mmae_test output was generated with the Benign0 data set.

Filter-Test Output — C++ Implementation

```
INITIALIZING FOGMA FILTER . . .

Filter successfully initialized


READING PATH FILE . . .

Path file ../PATHS/benign0.xyz read and normalized


BEGINNING SIMULATION LOOP . . .

***** Time = 1 *****

Updating for Z = [ -0.0597    0.9910   -0.1196 ](T)

Updated X = [ -0.0108    0.9984   -0.0217 ](T)
LQuot ()   = 1.467929
MQuot ()   = 0.017940
f          = 22.557404


Propagating one sample period ahead

Propagated X = [ -0.0119    0.9982   -0.0239 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0131    0.9980   -0.0262 ](T)
```

```
***** Time = 2 *****

Updating for Z = [ -0.0597   0.9909  -0.1206 ](T)

Updated X = [ -0.0206   0.9969  -0.0415 ](T)
LQuot ()  = 0.954828
MQuot ()  = 0.011669
f         = 29.154612


Propagating one sample period ahead

Propagated X = [ -0.0226   0.9966  -0.0456 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0247   0.9963  -0.0498 ](T)


***** Time = 3 *****

Updating for Z = [ -0.0596   0.9908  -0.1213 ](T)

Updated X = [ -0.0294   0.9955  -0.0593 ](T)
LQuot ()  = 0.584525
MQuot ()  = 0.007144
f         = 35.084595


Propagating one sample period ahead

Propagated X = [ -0.0321   0.9951  -0.0649 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0350   0.9947  -0.0707 ](T)


***** Time = 4 *****

Updating for Z = [ -0.0592   0.9908  -0.1220 ](T)

Updated X = [ -0.0371   0.9943  -0.0753 ](T)
LQuot ()  = 0.328595
MQuot ()  = 0.004016
f         = 39.873981
```

```
Propagating one sample period ahead

Propagated X = [ -0.0404    0.9938   -0.0821 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0438    0.9933   -0.0890 ](T)


***** Time = 5 *****

Updating for Z = [ -0.0589    0.9907   -0.1228 ](T)

Updated X = [ -0.0438    0.9932   -0.0895 ](T)
LQuot ()  = 0.164408
MQuot ()  = 0.002009
f         = 43.285347


Propagating one sample period ahead

Propagated X = [ -0.0475    0.9927   -0.0972 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0514    0.9921   -0.1050 ](T)


***** Time = 6 *****

Updating for Z = [ -0.0586    0.9906   -0.1235 ](T)

Updated X = [ -0.0495    0.9923   -0.1020 ](T)
LQuot ()  = 0.067255
MQuot ()  = 0.000822
f         = 45.439762


Propagating one sample period ahead

Propagated X = [ -0.0536    0.9916   -0.1103 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0577    0.9910   -0.1188 ](T)
```

```
***** Time = 7 *****

Updating for Z = [ -0.0582   0.9906  -0.1236 ](T)

Updated X = [ -0.0544   0.9915  -0.1127 ](T)
LQuot ()  = 0.016174
MQuot ()  = 0.000198
f         = 46.615192


Propagating one sample period ahead

Propagated X = [ -0.0586   0.9908  -0.1215 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0629   0.9901  -0.1304 ](T)


***** Time = 8 *****

Updating for Z = [ -0.0581   0.9907  -0.1231 ](T)

Updated X = [ -0.0585   0.9908  -0.1218 ](T)
LQuot ()  = 0.000232
MQuot ()  = 0.000003
f         = 46.988255


Propagating one sample period ahead

Propagated X = [ -0.0628   0.9901  -0.1307 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0672   0.9894  -0.1398 ](T)


***** Time = 9 *****

Updating for Z = [ -0.0577   0.9908  -0.1223 ](T)

Updated X = [ -0.0619   0.9902  -0.1292 ](T)
LQuot ()  = 0.008071
MQuot ()  = 0.000099
f         = 46.804520
```

```
Propagating one sample period ahead

Propagated X = [ -0.0661    0.9896   -0.1381 ](T)

Projecting two sample periods ahead

Projected X = [ -0.0704    0.9889   -0.1472 ](T)


***** Time = 10 *****

Updating for Z = [ -0.0572    0.9910   -0.1207 ](T)

Updated X = [ -0.0645    0.9898   -0.1350 ](T)
LQuot ()  = 0.031605
MQuot ()  = 0.000386
f         = 46.257156

Propagating one sample period ahead

Propagated X = [ -0.0687    0.9892   -0.1437 ](T)

Projecting two sample periods ahead

Projected X = [ -0.0728    0.9885   -0.1526 ](T)
```

Filter-Test Output — Ada 9XImplementation

```
INITIALIZING FOGMA FILTER . . .

Filter successfully initialized

READING PATH FILE . . .

Path file ../THESIS-C/PATHS/benign0.xyz read and normalized

BEGINNING SIMULATION LOOP . . .

***** Time =   1 *****

Updating for Z = [ -0.0597    0.9910   -0.1196 ](T)

Updated X = [ -0.0108    0.9984   -0.0217 ](T)
```

```
LQuot ()   =   1.467929
MQuot ()   =   0.017940
f          =  22.557405

Propagating one sample period ahead

Propagated X = [ -0.0119    0.9982   -0.0239 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0131    0.9980   -0.0262 ](T)


***** Time =   2 *****

Updating for Z = [ -0.0597    0.9909   -0.1206 ](T)

Updated X = [ -0.0206    0.9969   -0.0415 ](T)
LQuot ()   =   0.954828
MQuot ()   =   0.011669
f          =  29.154610

Propagating one sample period ahead

Propagated X = [ -0.0226    0.9966   -0.0456 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0247    0.9963   -0.0498 ](T)


***** Time =   3 *****

Updating for Z = [ -0.0596    0.9908   -0.1213 ](T)

Updated X = [ -0.0294    0.9955   -0.0593 ](T)
LQuot ()   =   0.584525
MQuot ()   =   0.007144
f          =  35.084595

Propagating one sample period ahead

Propagated X = [ -0.0321    0.9951   -0.0649 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0350    0.9947   -0.0707 ](T)
```

```
***** Time =   4 *****

Updating for Z = [ -0.0592   0.9908  -0.1220 ](T)

Updated X = [ -0.0371   0.9943  -0.0753 ](T)
LQuot ()  =   0.328595
MQuot ()  =   0.004016
f         =  39.873985

Propagating one sample period ahead

Propagated X = [ -0.0404   0.9938  -0.0821 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0438   0.9933  -0.0890 ](T)


***** Time =   5 *****

Updating for Z = [ -0.0589   0.9907  -0.1228 ](T)

Updated X = [ -0.0438   0.9932  -0.0895 ](T)
LQuot ()  =   0.164408
MQuot ()  =   0.002009
f         =  43.285347

Propagating one sample period ahead

Propagated X = [ -0.0475   0.9927  -0.0972 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0514   0.9921  -0.1050 ](T)


***** Time =   6 *****

Updating for Z = [ -0.0586   0.9906  -0.1235 ](T)

Updated X = [ -0.0495   0.9923  -0.1020 ](T)
LQuot ()  =   0.067255
MQuot ()  =   0.000822
f         =  45.439770
```

```
Propagating one sample period ahead

Propagated X = [ -0.0536    0.9916   -0.1103 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0577    0.9910   -0.1188 ](T)


***** Time =    7 *****

Updating for Z = [ -0.0582    0.9906   -0.1236 ](T)

Updated X = [ -0.0544    0.9915   -0.1127 ](T)
LQuot ()  =    0.016174
MQuot ()  =    0.000198
f         =   46.615185

Propagating one sample period ahead

Propagated X = [ -0.0586    0.9908   -0.1215 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0629    0.9901   -0.1304 ](T)


***** Time =    8 *****

Updating for Z = [ -0.0581    0.9907   -0.1231 ](T)

Updated X = [ -0.0585    0.9908   -0.1218 ](T)
LQuot ()  =    0.000232
MQuot ()  =    0.000003
f         =   46.988251

Propagating one sample period ahead

Propagated X = [ -0.0628    0.9901   -0.1307 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0672    0.9894   -0.1398 ](T)


***** Time =    9 *****
```

```
Updating for Z = [ -0.0577    0.9908   -0.1223 ](T)

Updated X = [ -0.0619    0.9902   -0.1292 ](T)
LQuot ()  =    0.008071
MQuot ()  =    0.000099
f         =   46.804520

Propagating one sample period ahead

Propagated X = [ -0.0661    0.9896   -0.1381 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0704    0.9889   -0.1472 ](T)


***** Time =  10 *****

Updating for Z = [ -0.0572    0.9910   -0.1207 ](T)

Updated X = [ -0.0645    0.9898   -0.1350 ](T)
LQuot ()  =    0.031605
MQuot ()  =    0.000386
f         =   46.257153

Propagating one sample period ahead

Propagated X = [ -0.0687    0.9892   -0.1437 ](T)


Projecting two sample periods ahead

Projected X = [ -0.0728    0.9885   -0.1526 ](T)
```

MMAE-Test Output -- C++ Implementation

```
INITIALIZING MMAE . . .

MMAE successfully initialized


READING PATH FILE . . .

Path file ../PATHS/benign0.xyz read and normalized


BEGINNING SIMULATION LOOP . . .
```

***** Time = 1 *****

Correcting for Z = [-0.0597 0.9910 -0.1196](T)

		Xp	residual	RAR	RR
		-0.0108	-0.0597		
Filter	1	0.9984	-0.0090	1.4679	0.0179
		-0.0217	-0.1196		
		-0.0265	-0.0597		
Filter	2	0.9960	-0.0090	0.9969	0.0179
		-0.0531	-0.1196		
		-0.0373	-0.0597		
Filter	3	0.9944	-0.0090	0.6730	0.0179
		-0.0747	-0.1196		

Corrected X = [-0.0216 0.9968 -0.0433](T)

Predicting one sample period ahead

Filter 1 Xm = [-0.0119 0.9982 -0.0239](T) pk1 = 0.46
Filter 2 Xm = [-0.0345 0.9948 -0.0691](T) pk2 = 0.33
Filter 3 Xm = [-0.0553 0.9917 -0.1108](T) pk3 = 0.21

Predicted X = [-0.0285 0.9957 -0.0572](T)

Extending two sample periods ahead

Extended X = [-0.0356 0.9946 -0.0714](T)

***** Time = 2 *****

Correcting for Z = [-0.0597 0.9909 -0.1206](T)

		Xp	residual	RAR	RR
		-0.0206	-0.0478		
Filter	1	0.9969	-0.0073	0.9548	0.0117
		-0.0415	-0.0966		
		-0.0457	-0.0252		
Filter	2	0.9931	-0.0039	0.1835	0.0033
		-0.0920	-0.0515		
		-0.0580	-0.0044		

```
Filter   3    0.9912        -0.0008         0.0043       0.0001
             -0.1169        -0.0098

Corrected X = [ -0.0334   0.9949  -0.0673 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0226   0.9966  -0.0456 ](T)   pk1 = 0.55
Filter 2 Xm = [ -0.0574   0.9913  -0.1155 ](T)   pk2 = 0.32
Filter 3 Xm = [ -0.0776   0.9882  -0.1564 ](T)   pk3 = 0.13

Predicted X = [ -0.0408   0.9938  -0.0822 ](T)


Extending two sample periods ahead

Extended X = [ -0.0483   0.9927  -0.0973 ](T)


***** Time = 3 *****

Correcting for Z = [ -0.0596   0.9908  -0.1213 ](T)

                Xp          residual         RAR           RR

             -0.0294        -0.0370
Filter   1    0.9955        -0.0058         0.5845       0.0071
             -0.0593        -0.0758

             -0.0584        -0.0023
Filter   2    0.9911        -0.0005         0.0022       0.0000
             -0.1181        -0.0058

             -0.0664         0.0180
Filter   3    0.9898         0.0026         0.0585       0.0016
             -0.1345         0.0351

Corrected X = [ -0.0399   0.9939  -0.0807 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0321   0.9951  -0.0649 ](T)   pk1 = 0.65
Filter 2 Xm = [ -0.0708   0.9892  -0.1432 ](T)   pk2 = 0.29
Filter 3 Xm = [ -0.0806   0.9876  -0.1635 ](T)   pk3 = 0.06

Predicted X = [ -0.0461   0.9930  -0.0933 ](T)


Extending two sample periods ahead
```

```
Extended X = [ -0.0524    0.9920   -0.1061 ](T)


***** Time = 4 *****

Correcting for Z = [ -0.0592    0.9908   -0.1220 ](T)

                  Xp          residual         RAR           RR

                -0.0371        -0.0271
Filter    1      0.9943        -0.0043         0.3286        0.0040
                -0.0753        -0.0571

                -0.0656         0.0115
Filter    2      0.9899         0.0016         0.0324        0.0006
                -0.1338         0.0212

                -0.0672         0.0213
Filter    3      0.9896         0.0031         0.0820        0.0022
                -0.1376         0.0415

Corrected X = [ -0.0440    0.9932   -0.0895 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0404    0.9938   -0.0821 ](T)    pk1 = 0.76
Filter 2 Xm = [ -0.0767    0.9882   -0.1566 ](T)    pk2 = 0.22
Filter 3 Xm = [ -0.0749    0.9883   -0.1540 ](T)    pk3 = 0.02

Predicted X = [ -0.0491    0.9925   -0.0999 ](T)


Extending two sample periods ahead

Extended X = [ -0.0543    0.9916   -0.1105 ](T)


***** Time = 5 *****

Correcting for Z = [ -0.0589    0.9907   -0.1228 ](T)

                  Xp          residual         RAR           RR

                -0.0438        -0.0186
Filter    1      0.9932        -0.0031         0.1644        0.0020
                -0.0895        -0.0407

                -0.0688         0.0178
```

```
Filter   2    0.9893          0.0025          0.0814        0.0015
             -0.1416         0.0338

             -0.0649         0.0160
Filter   3    0.9898          0.0023          0.0463        0.0012
             -0.1345         0.0312

Corrected X = [ -0.0475   0.9926  -0.0972 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0475   0.9927  -0.0972 ](T)   pk1 = 0.85
Filter 2 Xm = [ -0.0776   0.9879  -0.1600 ](T)   pk2 = 0.14
Filter 3 Xm = [ -0.0677   0.9893  -0.1413 ](T)   pk3 = 0.01

Predicted X = [ -0.0519   0.9919  -0.1064 ](T)


Extending two sample periods ahead

Extended X = [ -0.0565   0.9912  -0.1158 ](T)


***** Time = 6 *****

Correcting for Z = [ -0.0586   0.9906  -0.1235 ](T)

                  Xp          residual          RAR           RR

             -0.0495         -0.0111
Filter   1    0.9923         -0.0020          0.0673        0.0008
             -0.1020         -0.0263

             -0.0692          0.0190
Filter   2    0.9891          0.0027          0.0943        0.0017
             -0.1438          0.0365

             -0.0620          0.0091
Filter   3    0.9901          0.0013          0.0150        0.0004
             -0.1302          0.0177

Corrected X = [ -0.0513   0.9920  -0.1057 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0536   0.9916  -0.1103 ](T)   pk1 = 0.91
Filter 2 Xm = [ -0.0754   0.9881  -0.1573 ](T)   pk2 = 0.08
Filter 3 Xm = [ -0.0620   0.9900  -0.1315 ](T)   pk3 = 0.01
```

```
Predicted X = [ -0.0555    0.9913   -0.1144 ](T)

Extending two sample periods ahead

Extended X = [ -0.0597    0.9907   -0.1233 ](T)


***** Time = 7 *****

Correcting for Z = [ -0.0582    0.9906   -0.1236 ](T)

                  Xp          residual         RAR           RR

               -0.0544        -0.0046
Filter    1     0.9915        -0.0010         0.0162        0.0002
               -0.1127        -0.0132

               -0.0677         0.0172
Filter    2     0.9892         0.0026         0.0799        0.0014
               -0.1423         0.0337

               -0.0596         0.0038
Filter    3     0.9904         0.0006         0.0029        0.0001
               -0.1265         0.0079

Corrected X = [ -0.0551    0.9913   -0.1142 ](T)

Predicting one sample period ahead

Filter 1 Xm = [ -0.0586    0.9908   -0.1215 ](T)    pk1 = 0.94
Filter 2 Xm = [ -0.0715    0.9885   -0.1510 ](T)    pk2 = 0.05
Filter 3 Xm = [ -0.0584    0.9905   -0.1253 ](T)    pk3 = 0.01

Predicted X = [ -0.0592    0.9907   -0.1229 ](T)

Extending two sample periods ahead

Extended X = [ -0.0634    0.9900   -0.1317 ](T)


***** Time = 8 *****

Correcting for Z = [ -0.0581    0.9907   -0.1231 ](T)

                  Xp          residual         RAR           RR

               -0.0585         0.0005
```

```
Filter   1    0.9908        -0.0001         0.0002        0.0000
             -0.1218        -0.0016

             -0.0656         0.0134
Filter   2    0.9895         0.0022         0.0534        0.0010
             -0.1386         0.0279

             -0.0582         0.0003
Filter   3    0.9906         0.0002         0.0002        0.0000
             -0.1239         0.0022

Corrected X = [ -0.0587   0.9907  -0.1222 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0628   0.9901  -0.1307 ](T)   pk1 = 0.96
Filter 2 Xm = [ -0.0673   0.9891  -0.1432 ](T)   pk2 = 0.03
Filter 3 Xm = [ -0.0569   0.9908  -0.1220 ](T)   pk3 = 0.01

Predicted X = [ -0.0629   0.9901  -0.1310 ](T)


Extending two sample periods ahead

Extended X = [ -0.0671   0.9894  -0.1398 ](T)


***** Time = 9 *****

Correcting for Z = [ -0.0577   0.9908  -0.1223 ](T)

                  Xp          residual         RAR           RR

             -0.0619         0.0051
Filter   1    0.9902         0.0007         0.0081        0.0001
             -0.1292         0.0085

             -0.0630         0.0096
Filter   2    0.9899         0.0017         0.0296        0.0005
             -0.1339         0.0209

             -0.0574        -0.0008
Filter   3    0.9908         0.0001         0.0000        0.0000
             -0.1222        -0.0002

Corrected X = [ -0.0619   0.9902  -0.1292 ](T)


Predicting one sample period ahead
```

```
Filter 1 Xm = [ -0.0661    0.9896   -0.1381 ](T)    pk1 = 0.98
Filter 2 Xm = [ -0.0633    0.9898   -0.1353 ](T)    pk2 = 0.01
Filter 3 Xm = [ -0.0563    0.9910   -0.1203 ](T)    pk3 = 0.01

Predicted X = [ -0.0660    0.9896   -0.1379 ](T)


Extending two sample periods ahead

Extended X = [ -0.0702    0.9889   -0.1468 ](T)


***** Time = 10 *****

Correcting for Z = [ -0.0572    0.9910   -0.1207 ](T)

                    Xp          residual          RAR           RR

                -0.0645          0.0089
Filter    1      0.9898          0.0015          0.0316        0.0004
                -0.1350          0.0174

                -0.0606          0.0061
Filter    2      0.9903          0.0012          0.0140        0.0003
                -0.1288          0.0146

                -0.0569         -0.0009
Filter    3      0.9910          0.0001          0.0000        0.0000
                -0.1205         -0.0004

Corrected X = [ -0.0644    0.9898   -0.1348 ](T)


Predicting one sample period ahead

Filter 1 Xm = [ -0.0687    0.9892   -0.1437 ](T)    pk1 = 0.98
Filter 2 Xm = [ -0.0599    0.9904   -0.1279 ](T)    pk2 = 0.01
Filter 3 Xm = [ -0.0561    0.9912   -0.1188 ](T)    pk3 = 0.01

Predicted X = [ -0.0684    0.9892   -0.1433 ](T)


Extending two sample periods ahead

Extended X = [ -0.0725    0.9886   -0.1519 ](T)
```

MMAE-Test Output -- Ada 9X Implementation

```
INITIALIZING MMAE . . .
```

```
Initializing Filter  1
Initializing Filter  2
Initializing Filter  3
MMAE successfully initialized


READING PATH FILE . . .

Path file ../THESIS-C/PATHS/benign0.xyz read and normalized


BEGINNING SIMULATION LOOP . . .

***** Time =   1 *****

Correcting for Z = [ -0.0597   0.9910  -0.1196 ](T)

                  Xp          residual         RAR           RR

                -0.0108       -0.0597
Filter   1       0.9984       -0.0090        1.4679        0.0179
                -0.0217       -0.1196

                -0.0265       -0.0597
Filter   2       0.9960       -0.0090        0.9969        0.0179
                -0.0531       -0.1196

                -0.0373       -0.0597
Filter   3       0.9944       -0.0090        0.6730        0.0179
                -0.0747       -0.1196

Corrected X = [ -0.0216   0.9968  -0.0433 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0119   0.9982  -0.0239 ](T)   pk1 = 0.46
Filter 2 = [ -0.0345   0.9948  -0.0691 ](T)   pk2 = 0.33
Filter 3 = [ -0.0553   0.9917  -0.1108 ](T)   pk3 = 0.21

Predicted X = [ -0.0285   0.9957  -0.0572 ](T)


Extending two sample periods ahead

Extended X = [ -0.0356   0.9946  -0.0714 ](T)


***** Time =   2 *****
```

```
Correcting for Z = [ -0.0597   0.9909  -0.1206 ](T)

                    Xp          residual          RAR          RR

                -0.0206       -0.0478
Filter   1       0.9969       -0.0073          0.9548       0.0117
                -0.0415       -0.0966

                -0.0457       -0.0252
Filter   2       0.9931       -0.0039          0.1835       0.0033
                -0.0920       -0.0515

                -0.0580       -0.0044
Filter   3       0.9912       -0.0008          0.0043       0.0001
                -0.1169       -0.0098

Corrected X = [ -0.0334   0.9949  -0.0673 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0226   0.9966  -0.0456 ](T)   pk1 = 0.55
Filter 2 = [ -0.0574   0.9913  -0.1155 ](T)   pk2 = 0.32
Filter 3 = [ -0.0776   0.9882  -0.1564 ](T)   pk3 = 0.13

Predicted X = [ -0.0408   0.9938  -0.0822 ](T)


Extending two sample periods ahead

Extended X = [ -0.0483   0.9927  -0.0973 ](T)


***** Time =   3 *****

Correcting for Z = [ -0.0596   0.9908  -0.1213 ](T)

                    Xp          residual          RAR          RR

                -0.0294       -0.0370
Filter   1       0.9955       -0.0058          0.5845       0.0071
                -0.0593       -0.0758

                -0.0584       -0.0023
Filter   2       0.9911       -0.0005          0.0022       0.0000
                -0.1181       -0.0058

                -0.0664        0.0180
Filter   3       0.9898        0.0026          0.0585       0.0016
                -0.1345        0.0351
```

```
Corrected X = [ -0.0399    0.9939   -0.0807 ](T)

Predicting one sample period ahead

Filter 1 = [ -0.0321    0.9951   -0.0649 ](T)    pk1 = 0.65
Filter 2 = [ -0.0708    0.9892   -0.1432 ](T)    pk2 = 0.29
Filter 3 = [ -0.0806    0.9876   -0.1635 ](T)    pk3 = 0.06

Predicted X = [ -0.0461    0.9930   -0.0933 ](T)

Extending two sample periods ahead

Extended X = [ -0.0524    0.9920   -0.1061 ](T)


***** Time =   4 *****

Correcting for Z = [ -0.0592    0.9908   -0.1220 ](T)

                    Xp          residual          RAR           RR

                 -0.0371       -0.0271
Filter   1        0.9943       -0.0043          0.3286        0.0040
                 -0.0753       -0.0571

                 -0.0656        0.0115
Filter   2        0.9899        0.0016          0.0324        0.0006
                 -0.1338        0.0212

                 -0.0672        0.0213
Filter   3        0.9896        0.0031          0.0820        0.0022
                 -0.1376        0.0415

Corrected X = [ -0.0440    0.9932   -0.0895 ](T)

Predicting one sample period ahead

Filter 1 = [ -0.0404    0.9938   -0.0821 ](T)    pk1 = 0.76
Filter 2 = [ -0.0767    0.9882   -0.1566 ](T)    pk2 = 0.22
Filter 3 = [ -0.0749    0.9883   -0.1540 ](T)    pk3 = 0.02

Predicted X = [ -0.0491    0.9925   -0.0999 ](T)

Extending two sample periods ahead

Extended X = [ -0.0543    0.9916   -0.1105 ](T)
```

***** Time = 5 *****

Correcting for Z = [-0.0589 0.9907 -0.1228](T)

		Xp	residual	RAR	RR
Filter	1	-0.0438 0.9932 -0.0895	-0.0186 -0.0031 -0.0407	0.1644	0.0020
Filter	2	-0.0688 0.9893 -0.1416	0.0178 0.0025 0.0338	0.0814	0.0015
Filter	3	-0.0649 0.9898 -0.1345	0.0160 0.0023 0.0312	0.0463	0.0012

Corrected X = [-0.0475 0.9926 -0.0972](T)

Predicting one sample period ahead

Filter 1 = [-0.0475 0.9927 -0.0972](T) pk1 = 0.85
Filter 2 = [-0.0776 0.9879 -0.1600](T) pk2 = 0.14
Filter 3 = [-0.0677 0.9893 -0.1413](T) pk3 = 0.01

Predicted X = [-0.0519 0.9919 -0.1064](T)

Extending two sample periods ahead

Extended X = [-0.0565 0.9912 -0.1158](T)

***** Time = 6 *****

Correcting for Z = [-0.0586 0.9906 -0.1235](T)

		Xp	residual	RAR	RR
Filter	1	-0.0495 0.9923 -0.1020	-0.0111 -0.0020 -0.0263	0.0673	0.0008
Filter	2	-0.0692 0.9891 -0.1438	0.0190 0.0027 0.0365	0.0943	0.0017

```
                -0.0620        0.0091
Filter    3      0.9901        0.0013        0.0150        0.0004
                -0.1302        0.0177

Corrected X = [ -0.0513    0.9920   -0.1057 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0536    0.9916   -0.1103 ](T)    pk1 = 0.91
Filter 2 = [ -0.0754    0.9881   -0.1573 ](T)    pk2 = 0.08
Filter 3 = [ -0.0620    0.9900   -0.1315 ](T)    pk3 = 0.01

Predicted X = [ -0.0555    0.9913   -0.1144 ](T)


Extending two sample periods ahead

Extended X = [ -0.0597    0.9907   -0.1233 ](T)


***** Time =    7 *****

Correcting for Z = [ -0.0582    0.9906   -0.1236 ](T)

                   Xp          residual         RAR           RR

                -0.0544        -0.0046
Filter    1      0.9915        -0.0010        0.0162        0.0002
                -0.1127        -0.0132

                -0.0677         0.0172
Filter    2      0.9892         0.0026        0.0799        0.0014
                -0.1423         0.0337

                -0.0596         0.0038
Filter    3      0.9904         0.0006        0.0029        0.0001
                -0.1265         0.0079

Corrected X = [ -0.0551    0.9913   -0.1142 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0586    0.9908   -0.1215 ](T)    pk1 = 0.94
Filter 2 = [ -0.0715    0.9885   -0.1510 ](T)    pk2 = 0.05
Filter 3 = [ -0.0584    0.9905   -0.1253 ](T)    pk3 = 0.01

Predicted X = [ -0.0592    0.9907   -0.1229 ](T)
```

Extending two sample periods ahead

Extended X = [-0.0634 0.9900 -0.1317](T)

***** Time = 8 *****

Correcting for Z = [-0.0581 0.9907 -0.1231](T)

		Xp	residual	RAR	RR
		-0.0585	0.0005		
Filter	1	0.9908	-0.0001	0.0002	0.0000
		-0.1218	-0.0016		
		-0.0656	0.0134		
Filter	2	0.9895	0.0022	0.0534	0.0010
		-0.1386	0.0279		
		-0.0582	0.0003		
Filter	3	0.9906	0.0002	0.0002	0.0000
		-0.1239	0.0022		

Corrected X = [-0.0587 0.9907 -0.1222](T)

Predicting one sample period ahead

Filter 1 = [-0.0628 0.9901 -0.1307](T) pk1 = 0.96
Filter 2 = [-0.0673 0.9891 -0.1432](T) pk2 = 0.03
Filter 3 = [-0.0569 0.9908 -0.1220](T) pk3 = 0.01

Predicted X = [-0.0629 0.9901 -0.1310](T)

Extending two sample periods ahead

Extended X = [-0.0671 0.9894 -0.1398](T)

***** Time = 9 *****

Correcting for Z = [-0.0577 0.9908 -0.1223](T)

		Xp	residual	RAR	RR
		-0.0619	0.0051		
Filter	1	0.9902	0.0007	0.0081	0.0001
		-0.1292	0.0085		

```
                -0.0630        0.0096
Filter    2      0.9899        0.0017        0.0296        0.0005
                -0.1339        0.0209

                -0.0574       -0.0008
Filter    3      0.9908        0.0001        0.0000        0.0000
                -0.1222       -0.0002

Corrected X = [ -0.0619    0.9902   -0.1292 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0661    0.9896   -0.1381 ](T)    pk1 = 0.98
Filter 2 = [ -0.0633    0.9898   -0.1353 ](T)    pk2 = 0.01
Filter 3 = [ -0.0563    0.9910   -0.1203 ](T)    pk3 = 0.01

Predicted X = [ -0.0660    0.9896   -0.1379 ](T)


Extending two sample periods ahead

Extended X = [ -0.0702    0.9889   -0.1468 ](T)


***** Time =  10 *****

Correcting for Z = [ -0.0572    0.9910   -0.1207 ](T)

                   Xp          residual         RAR           RR

                -0.0645        0.0089
Filter    1      0.9898        0.0015        0.0316        0.0004
                -0.1350        0.0174

                -0.0606        0.0061
Filter    2      0.9903        0.0012        0.0140        0.0003
                -0.1288        0.0146

                -0.0569       -0.0009
Filter    3      0.9910        0.0001        0.0000        0.0000
                -0.1205       -0.0004

Corrected X = [ -0.0644    0.9898   -0.1348 ](T)


Predicting one sample period ahead

Filter 1 = [ -0.0687    0.9892   -0.1437 ](T)    pk1 = 0.98
Filter 2 = [ -0.0599    0.9904   -0.1279 ](T)    pk2 = 0.01
Filter 3 = [ -0.0561    0.9912   -0.1188 ](T)    pk3 = 0.01
```

Predicted X = [-0.0684 0.9892 -0.1433](T)

Extending two sample periods ahead

Extended X = [-0.0725 0.9886 -0.1519](T)

Appendix C:
Performance Study Instructions

PURPOSE

This study is designed to compare the effectiveness of Polhemus tracking, single filter (non-adaptive) tracking, and adaptive Kalman filter tracking in a virtual environment. Specifically, the data collected will be used to determine if adaptive Kalman filtering offers any advantage over the other modes in reducing display lag. This is a theoretical research project, and is NOT a preliminary step in any USAF program. Further, the data collected in this experiment will not be used to evaluate individual performance.

FORMAT

This study will be conducted in the following manner. First, you will be given an overview of both the software and hardware to be used. Next, you will be given an opportunity to familiarize yourself with both the equipment and the virtual environment. You will then be asked to perform a series of trials. The objective of these trials is to keep a crosshair centered on a moving ball by changing your head orientation; in other words, follow the bouncing ball.

TRAINING

[Show the subject the Polhemus 3SPACE Tracker] This is the Polhemus 3SPACE Magnetic Tracker, or Polhemus for short. *[Show the subject the source element]* It uses three orthogonal magnetic coils located inside this source unit that

are independently pulsed and picked up by the sensor attached to the PT-01 HMD. This is how your head orientation is monitored.

[Show the subject the PT-01 HMD] This is the PT-01 Head Mounted Display Unit or HMD for short. It is similar to looking through a pair of binoculars. *[Show the subject the underside of the housing with the adjustment levers]* These levers allow you to adjust the interocular distance (the distance between the eyes) and the focus for each eye image. *[Show the subject the various position adjustment knobs]* The HMD housing can be adjusted in, out, up, and down for comfort, and the head band can also be tightened and/or loosened. Go ahead and put the HMD on, making certain it is adjusted comfortably. *[Allow the subject to try on the HMD. Help him/her adjust it "comfortably" onto his/her head. Load up the benign0 data set and set the tracking mode appropriately for this subject]* You should be able to see a green ground plane with a cross-hatch texture on it. This texture has no significance to the experiment except that it provides you with motion cues when you turn your head. You should also see a blue sky, a black crosshair in the middle of the screen, and a red ball somewhere near the crosshair. *[Make certain the subject can see all this]*

Each trial will be conducted in the same way. In order to start a trial, you simply hit the space bar. When you do, the crosshair in the center of the screen will turn white, indicating that the trial has started. Also, a box will appear around the crosshair. It will be either red or blue. Whatever color the box is, find the ball of the same color as quickly as you can and track its movement. This is what it will look like *[Hit the space bar to start the trial]*

Three types of trial will be used. The first type, which you are currently looking at, is benign motion. The ball makes very slow movements. *[Let the subject finish the trial; load up the moderate0 data set]* The second type is moderate movement. The ball makes very short, but very quick movements. I am not

expecting that you will be able to follow moderate motion perfectly; just do the best that you can. *[Let the subject watch the moderate trial; load up the mixed0 data set].* Also, for these two trials you may center the crosshair in the ball before you begin the trial if you like.

The final trial is a mixed trial. If you look to your left, you will see the red ball; if you look to your right, the blue ball. These balls will move about in the general area they are currently in, but the red ball will always be to your left, and the blue ball to your right. Remember, whatever the color of the box around the cursor is, find that ball AS QUICKLY AS YOU CAN and track it. If the box around the cursor should change color, find the other ball AS QUICKLY AS YOU CAN and track it. Since the initial color of the box around the cursor is random, I suggest you start by simply picking a point roughly between the two balls. This task roughly analogous to being in trail behind a friendly aircraft, and watching for enemy aircraft. You need to keep track of both as much as possible. *[Let the subject view the mixed trial].*

The experiment will be conducted as follows. You will do seven trials (two benign, two moderate, and three mixed) for a particular tracking mode. Then we will change the tracking mode and do the same seven trials (two benign, two moderate, and three mixed) again. Then we will switch tracking modes one more time and do the same seven trials again. Each of the trials is approximately thirty seconds in length, and you start each by hitting the space bar.

Trials

[Do the trials as prescribed by the data sheet. Be certain to tell the subject what type of trial he/she is doing, and also which tracking mode he/she is using. Be certain to use the appropriate tracking mode!!!]

Debrief

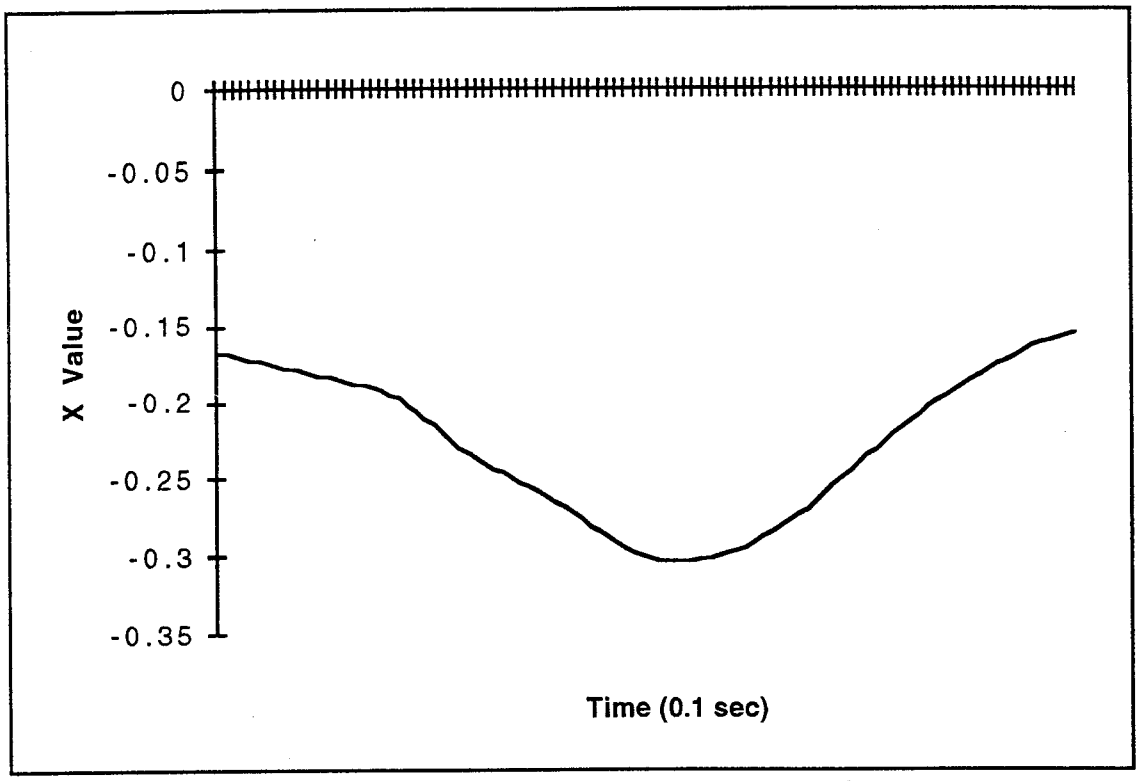
[When the trials are complete, ask the subject if he/she noted any differences between the various tracking modes. Also, ask if one seemed any easier or harder than the others.]

Appendix D:
Mean +/- 1 Sigma Plots

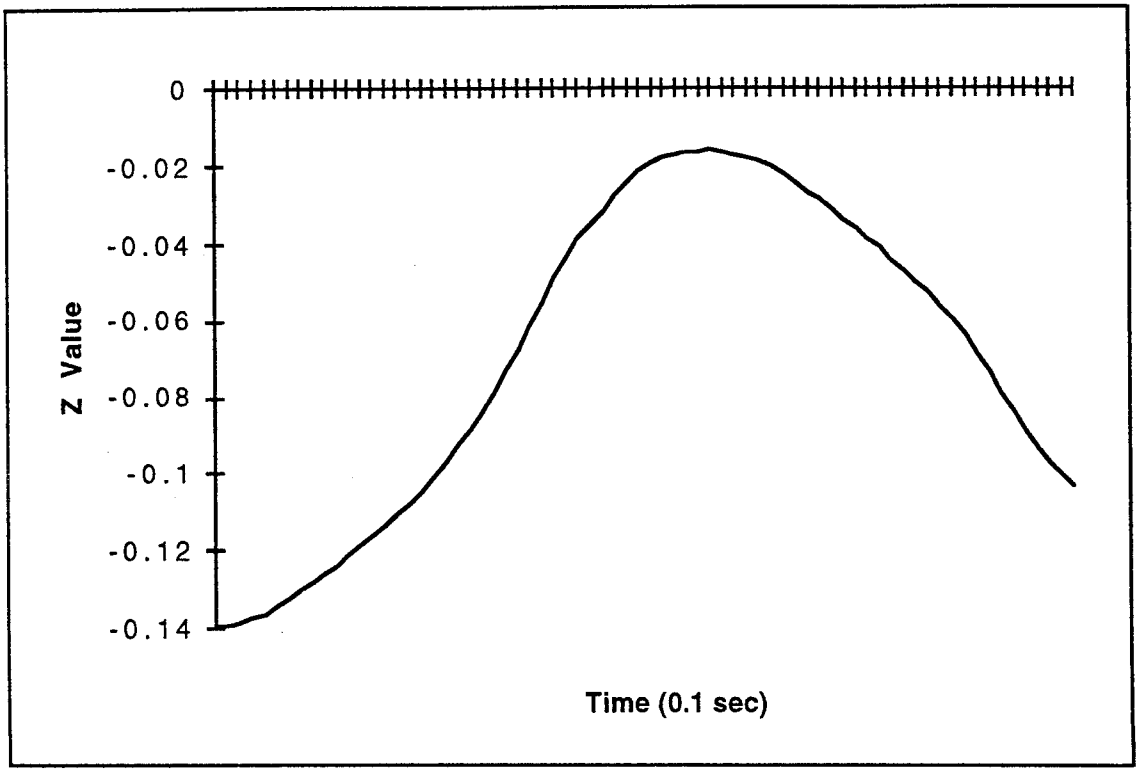
This appendix contains mean +/- sigma plots for selected data sets (Benign1, Moderate1, and Mixed1) from the data collected during the performance study. Only the middle ten seconds of data from a thirty-second data sample was plotted in these graphs due to space constraints. The graphs are arranged so that X- and Z-direction graphs for the same condition are on the same page.

In order to have something against which to compare prediction value, a *simulated Polhemus error* was defined as the difference between the Polhemus values at time t and the Polhemus values at time $t+1$; this definition accounts for the fact that under normal conditions, Polhemus data is one frame old when it is used to render a scene.

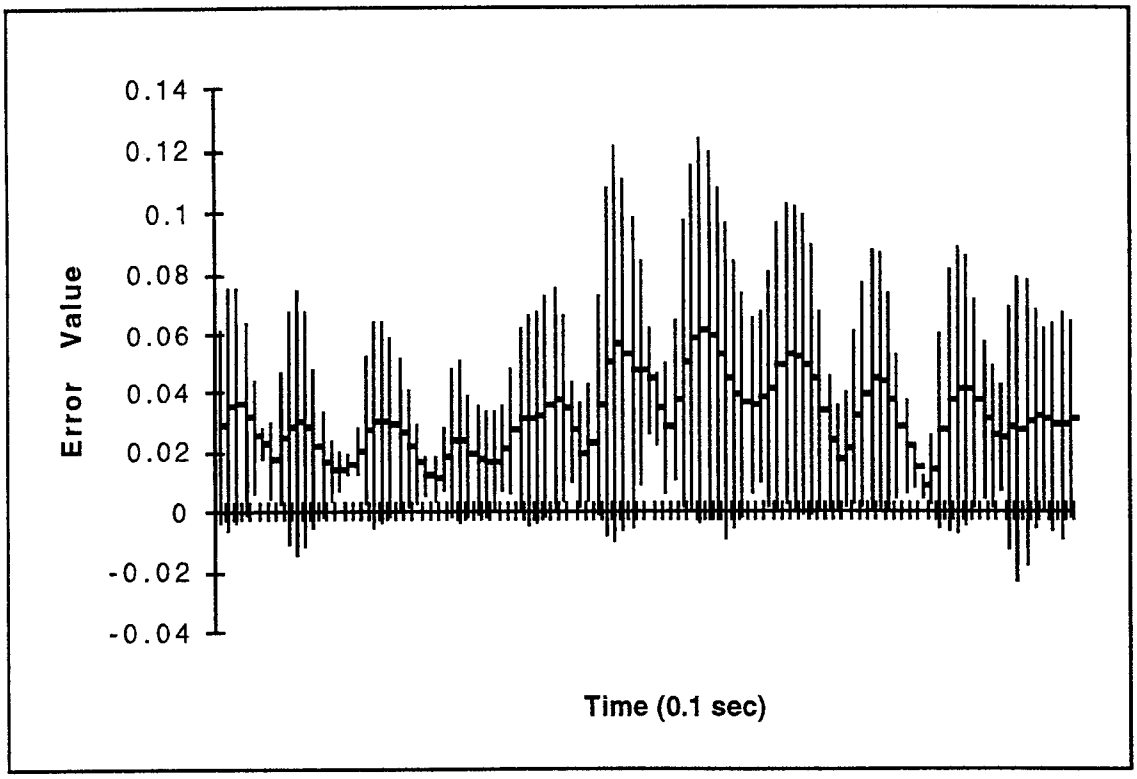
The graphs were generated by caluculating the absolute error value expressed as the difference between the predicted (or simulated predicted in the case of Polhemus Only mode) value, and the appropriate Polhemus input value, such that the result was always positive. These absolute errors were calculated for each subject and each sample period in the data set. The mean and standard deviation of these errors across all subjects for each sample period was calculated and used to produce the graphs in this appendix. Note that the graphs depicted are not for the same subjects in all conditions. The experiment was not designed such that a particular subject did all data sets in all tracking modes. However, in all cases, the graphs represent data for six subjects.



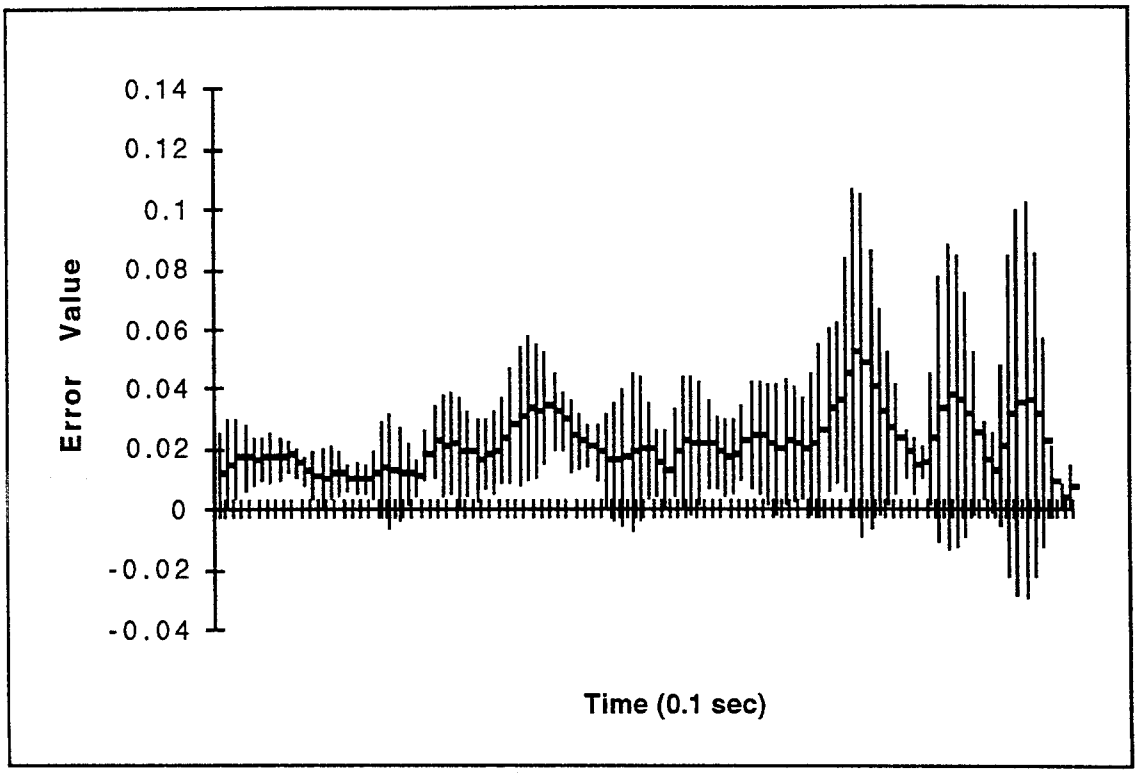
X-direction Path Values, Benign1 Data Set



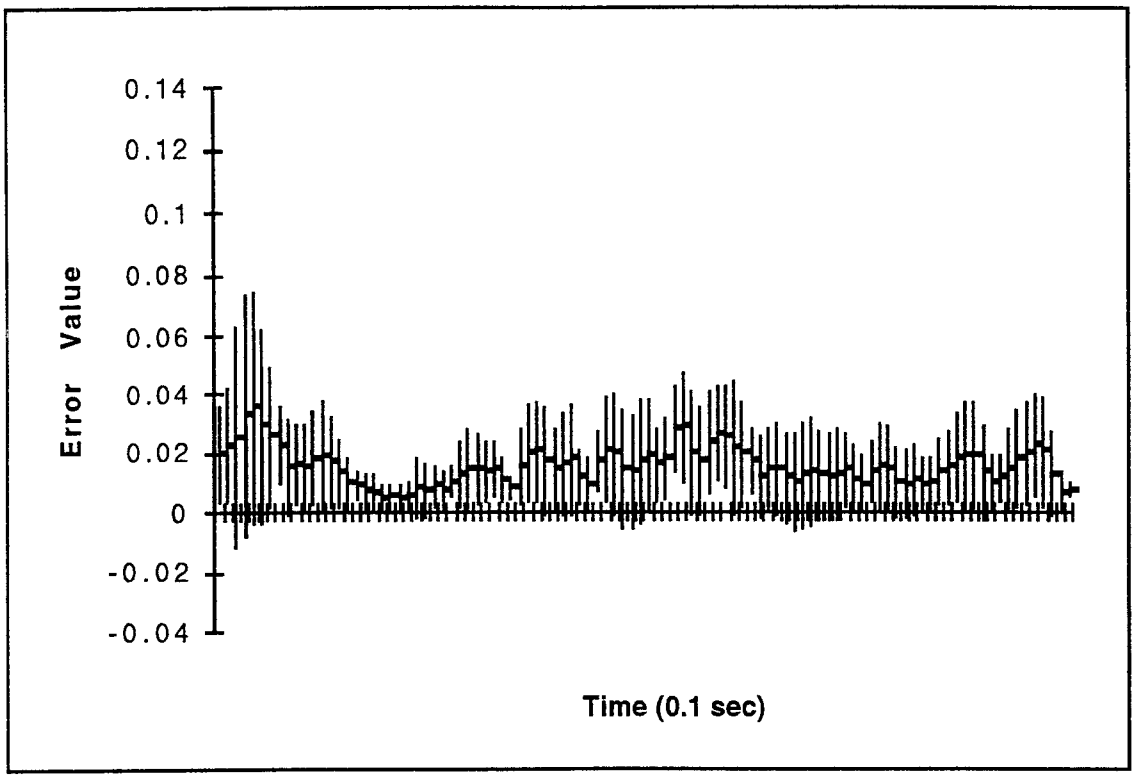
Z-direction Path Values, Benign1 Data Set



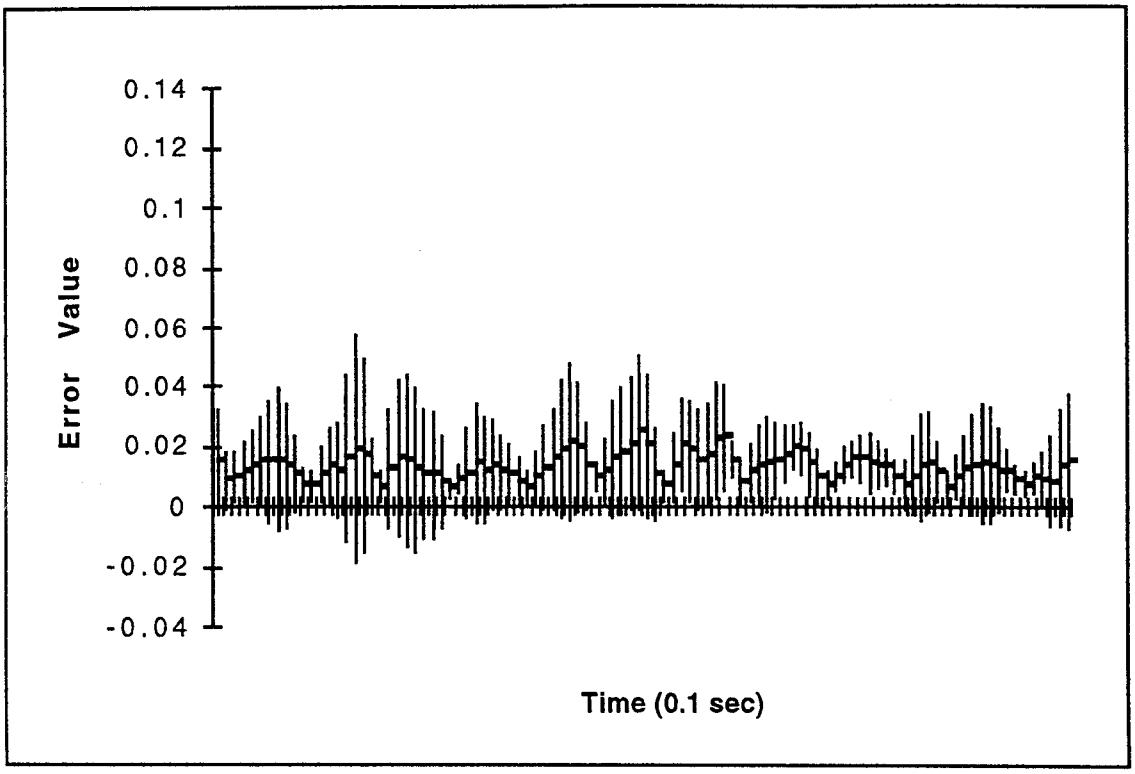
MMAE Prediction Error, X-direction, Benign1 Data Set



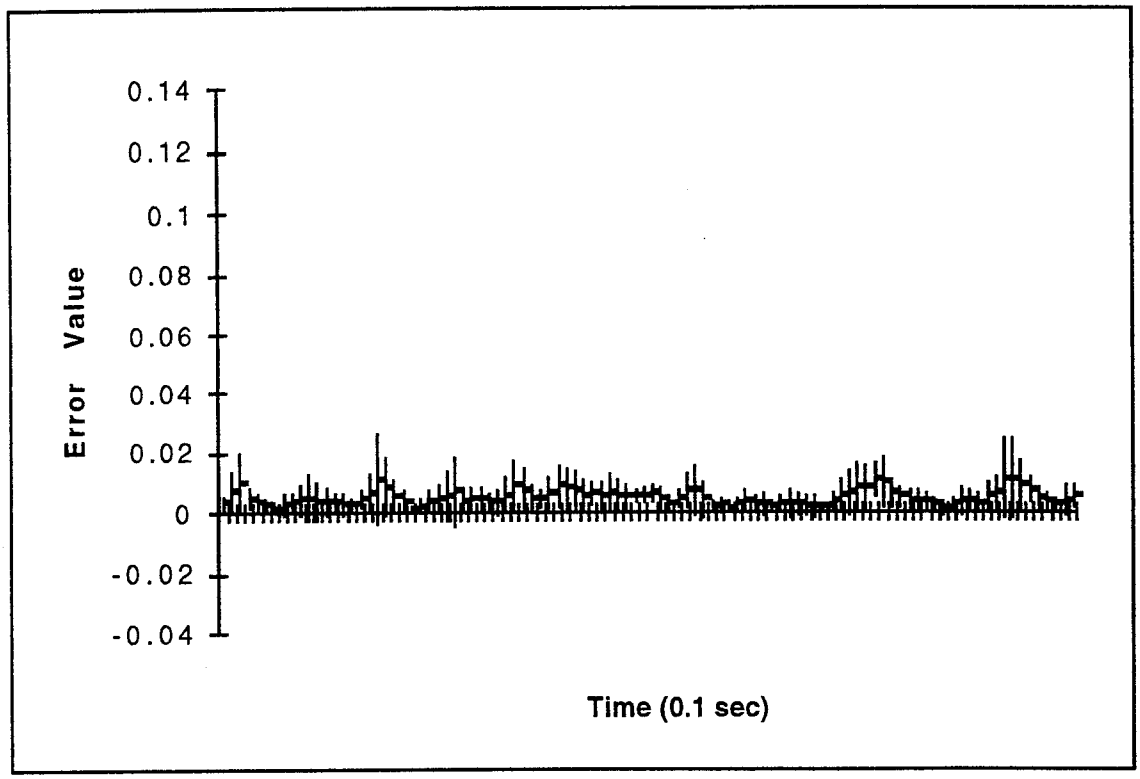
MMAE Prediction Error, Z-direction, Benign1 Data Set



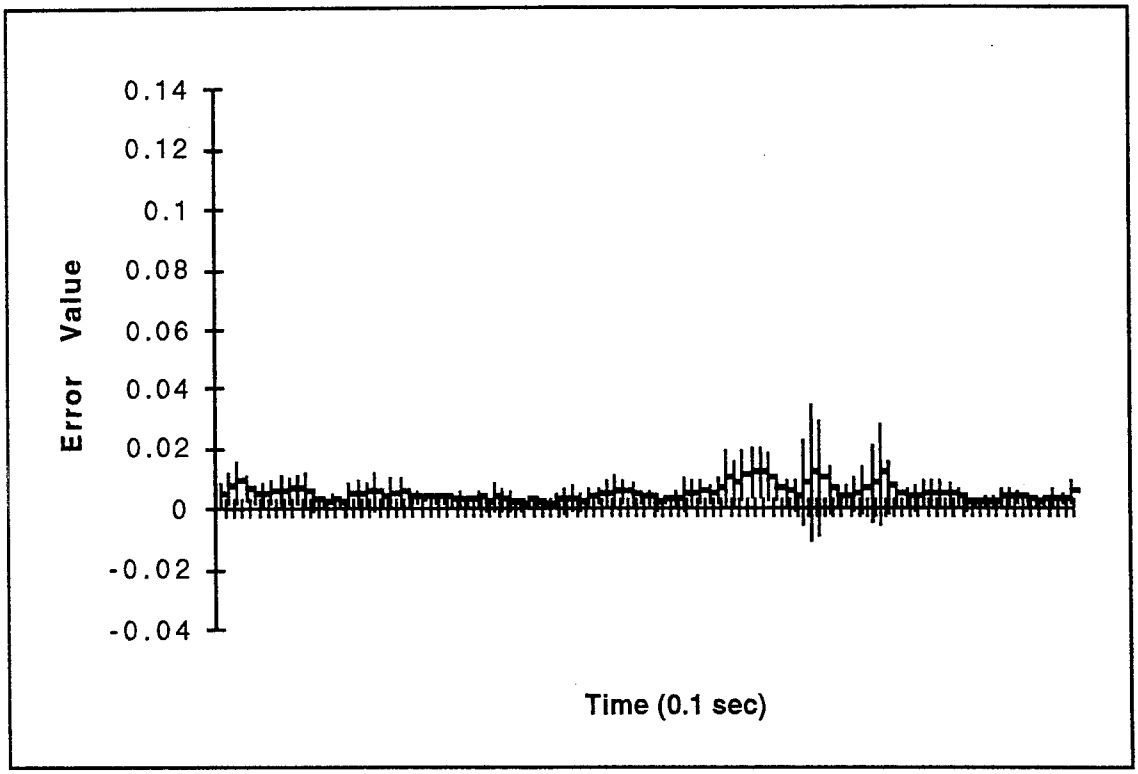
Single Filter Prediction Error, X-direction, Benign1 Data Set



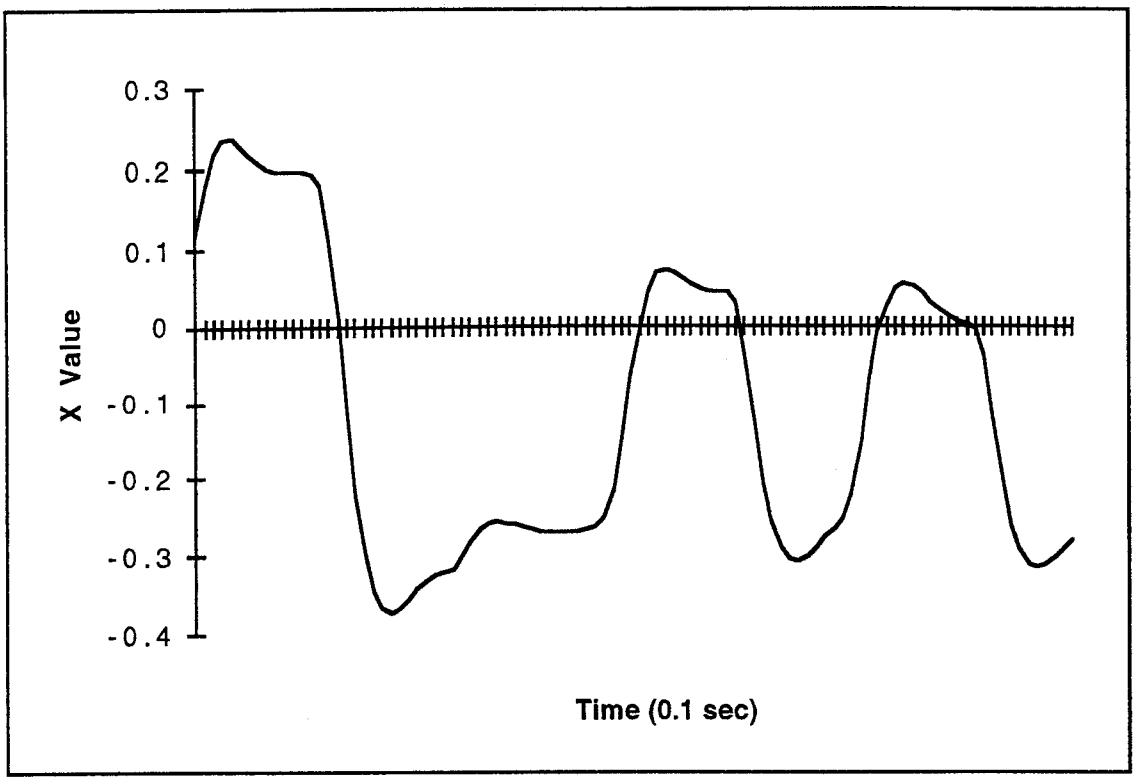
Single Filter Prediction Error, Z-direction, Benign1 Data Set



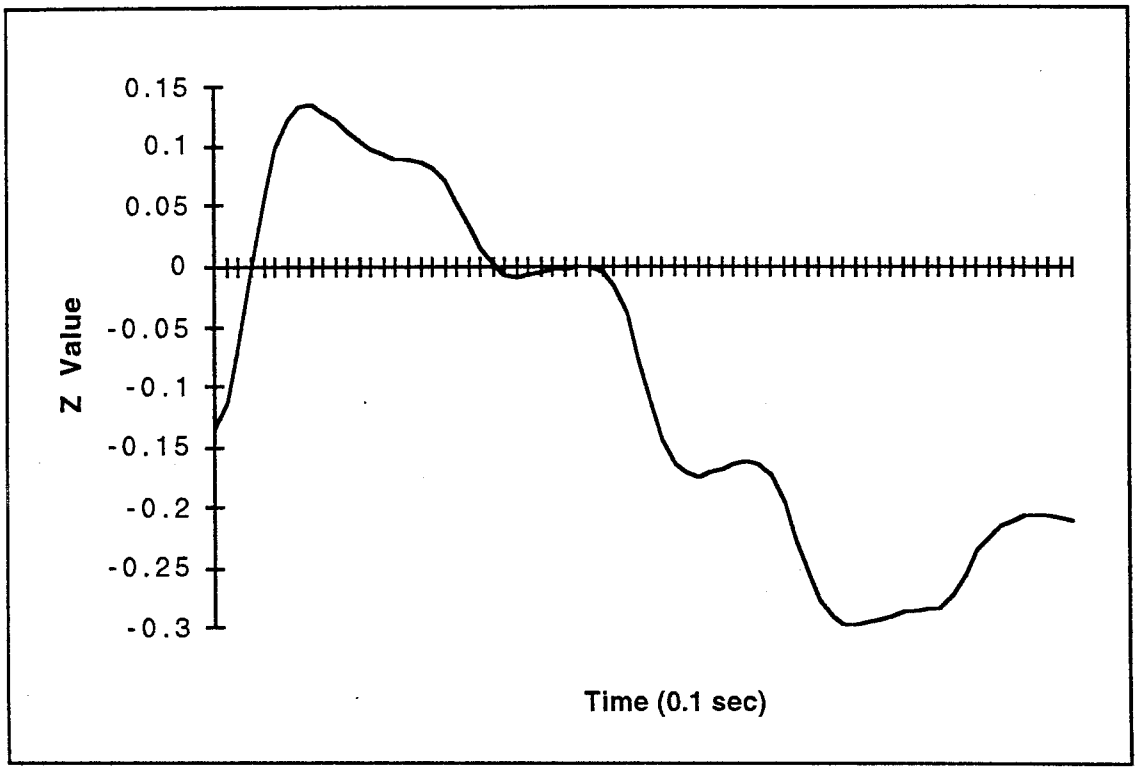
Simulated Polhemus Prediction Error, X-direction, Benign1 Data Set



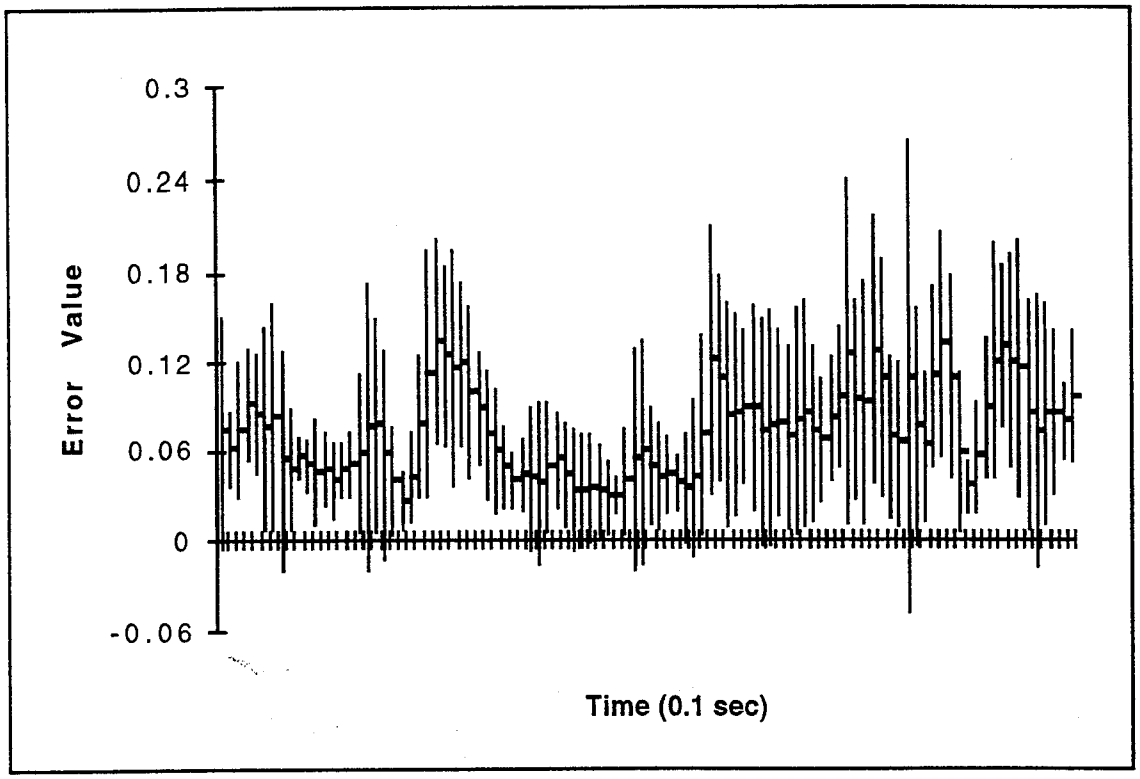
Simulated Polhemus Prediction Error, Z-direction, Benign1 Data Set



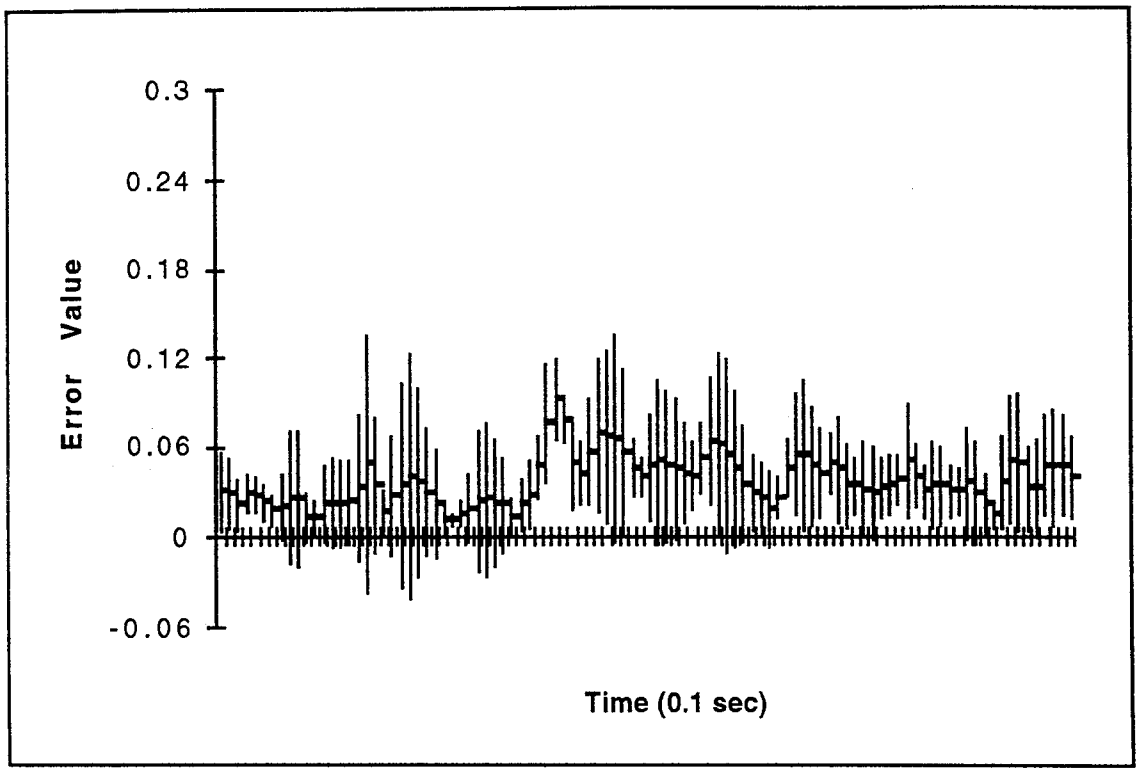
X-direction Path Values, Moderate1 Data Set



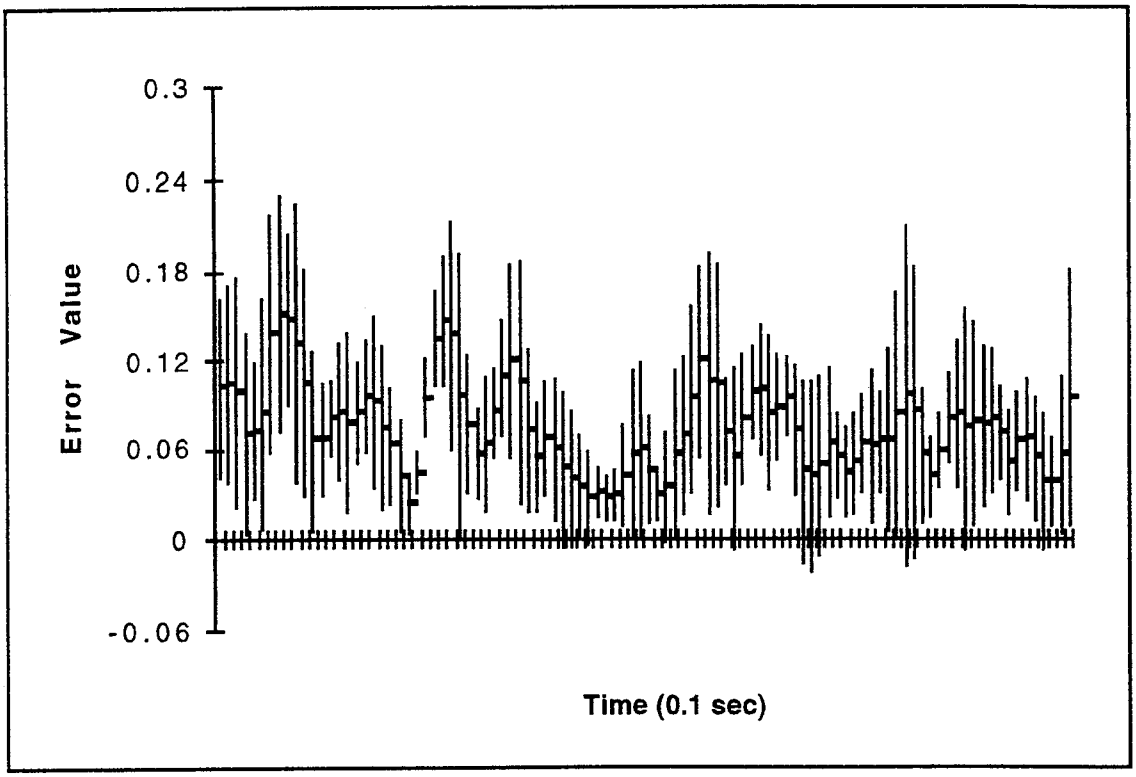
Z-direction Path Values, Moderate1 Data Set



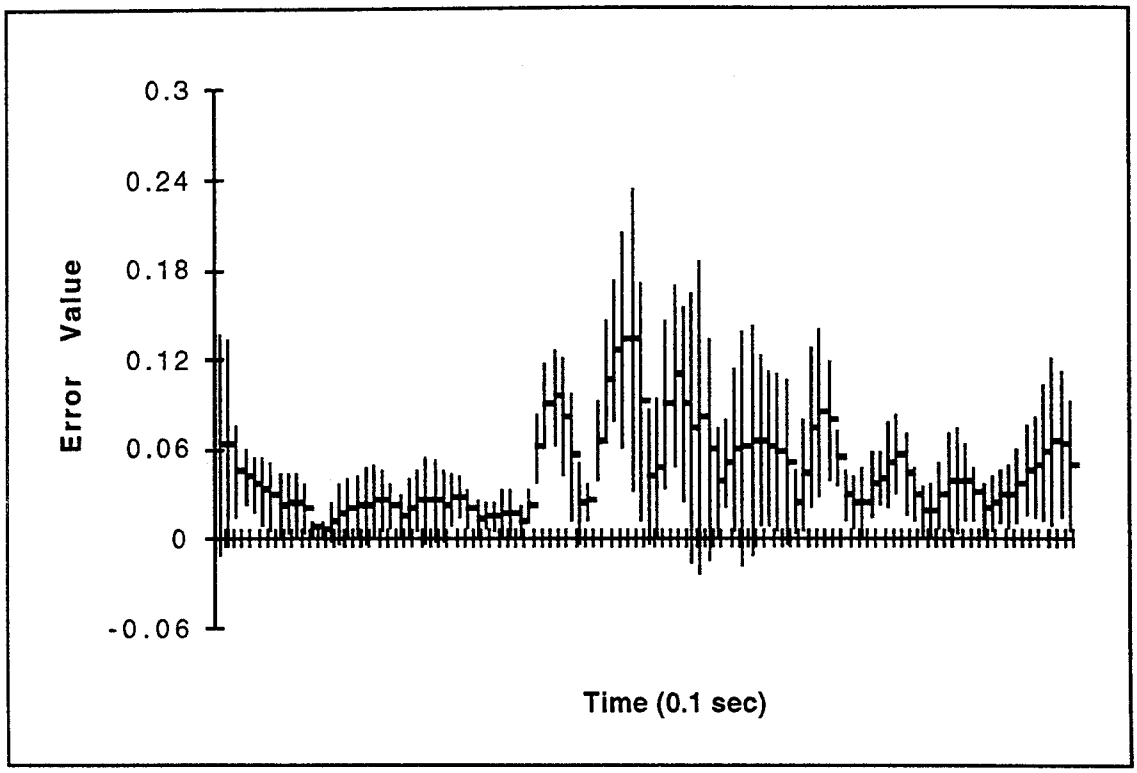
MMAE Prediction Error, X-direction Values, Moderate1 Data Set



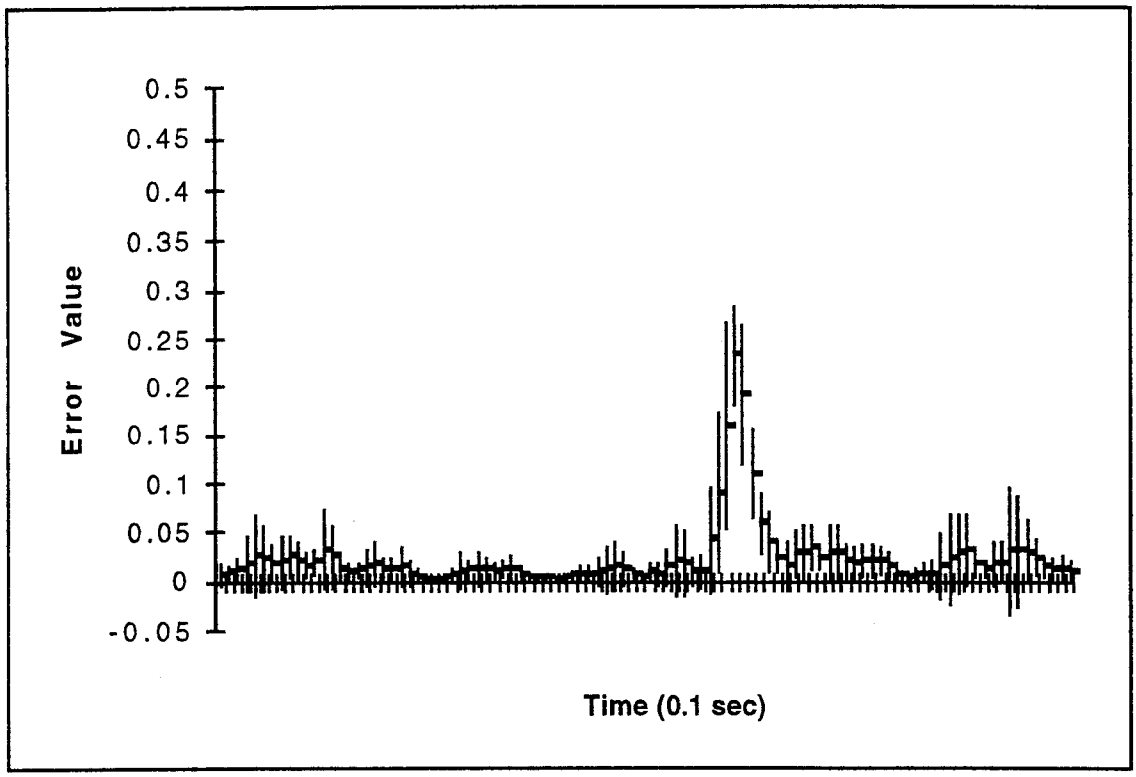
MMAE Prediction Error, Z-direction Values, Moderate1 Data Set



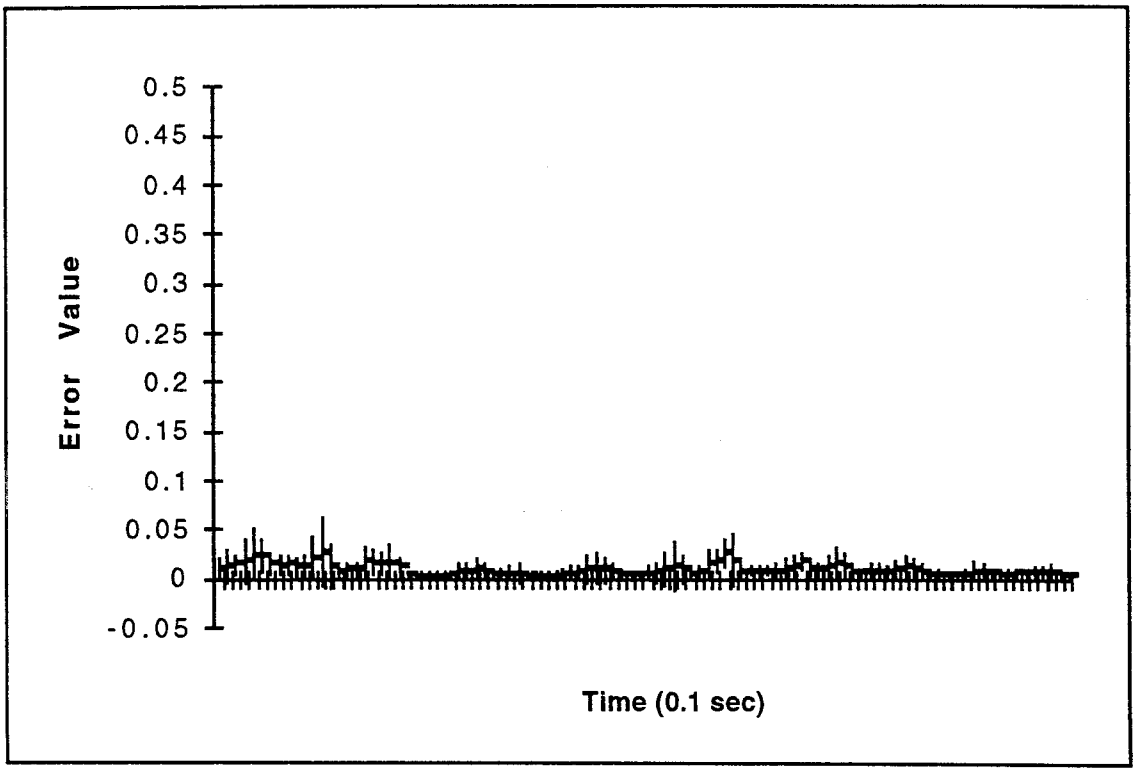
Single Filter Prediction Error, X-direction Values, Moderate1 Data Set



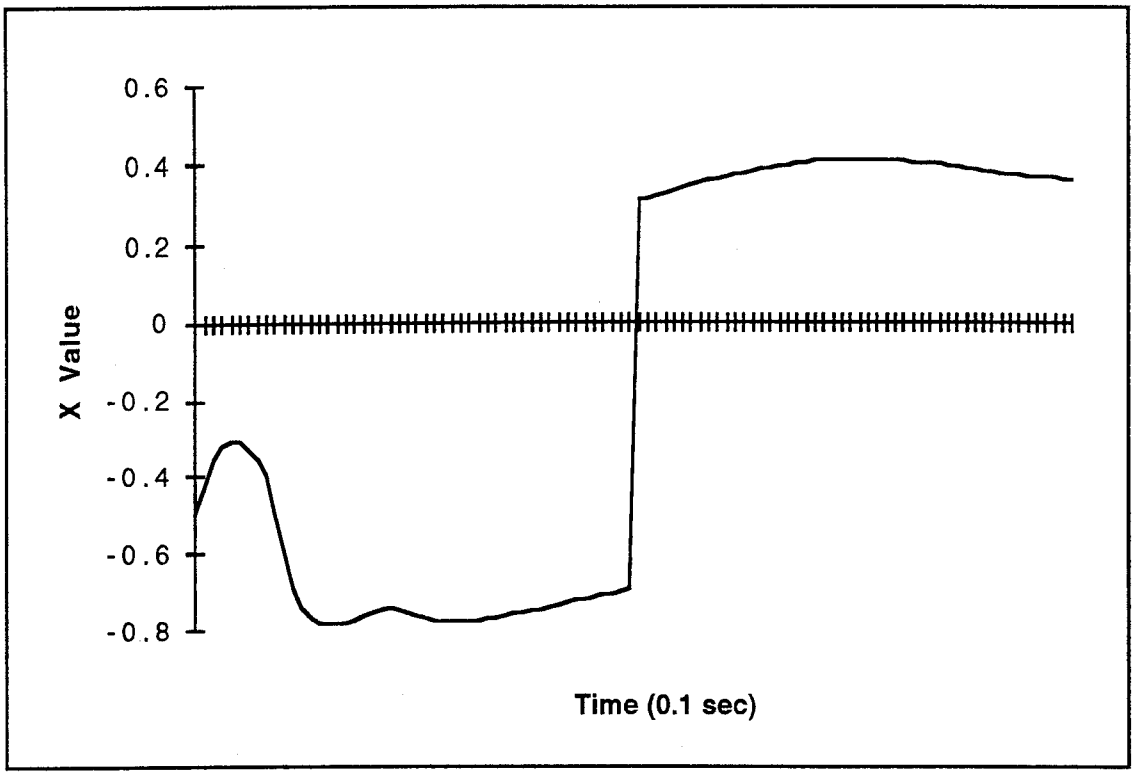
Single Filter Prediction Error, Z-direction Values, Moderate1 Data Set



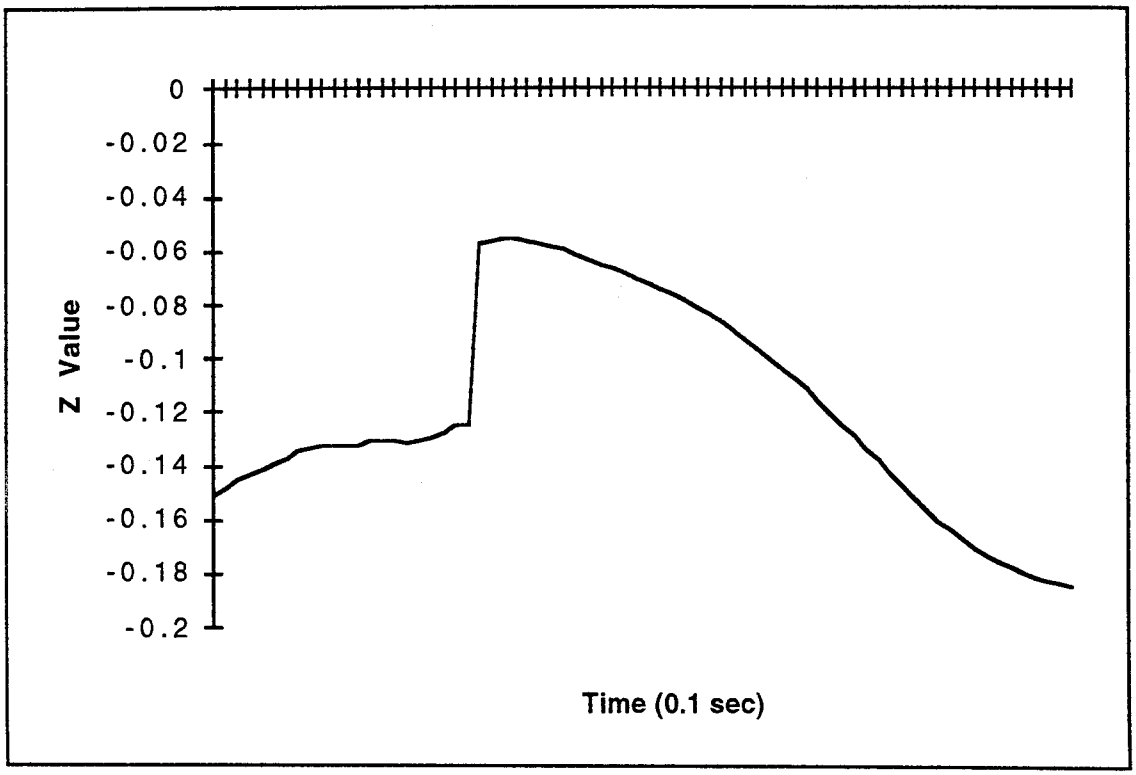
Simulated Polhemus Prediction Error, X-direction Values, Moderate1 Data Set



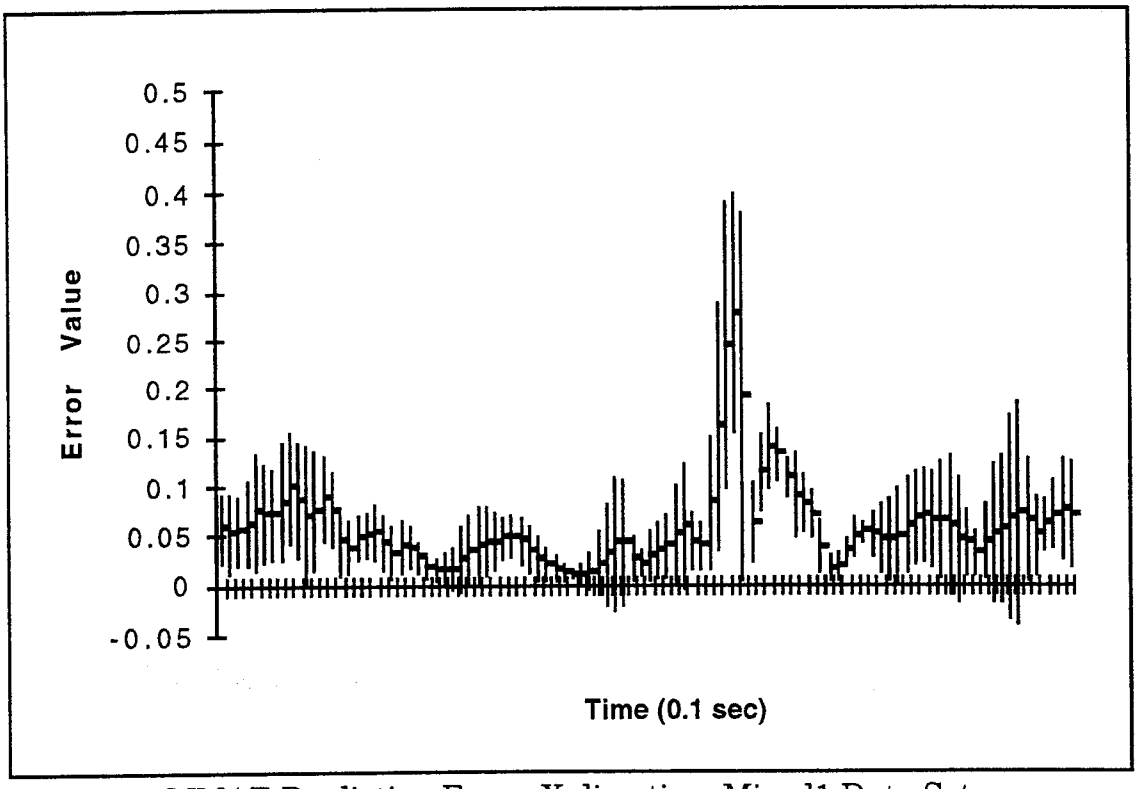
Simulated Polhemus Prediction Error, Z-direction Values, Moderate1 Data Set



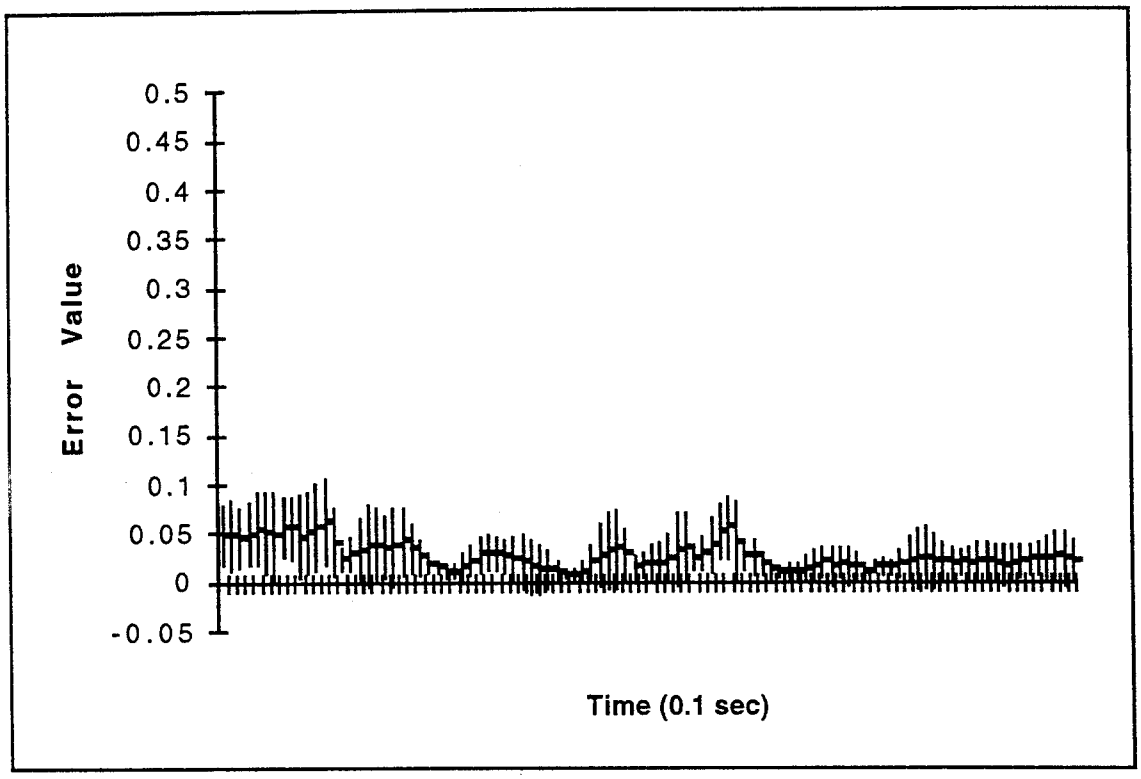
X-direction Path Values, Mixed1 Data Set



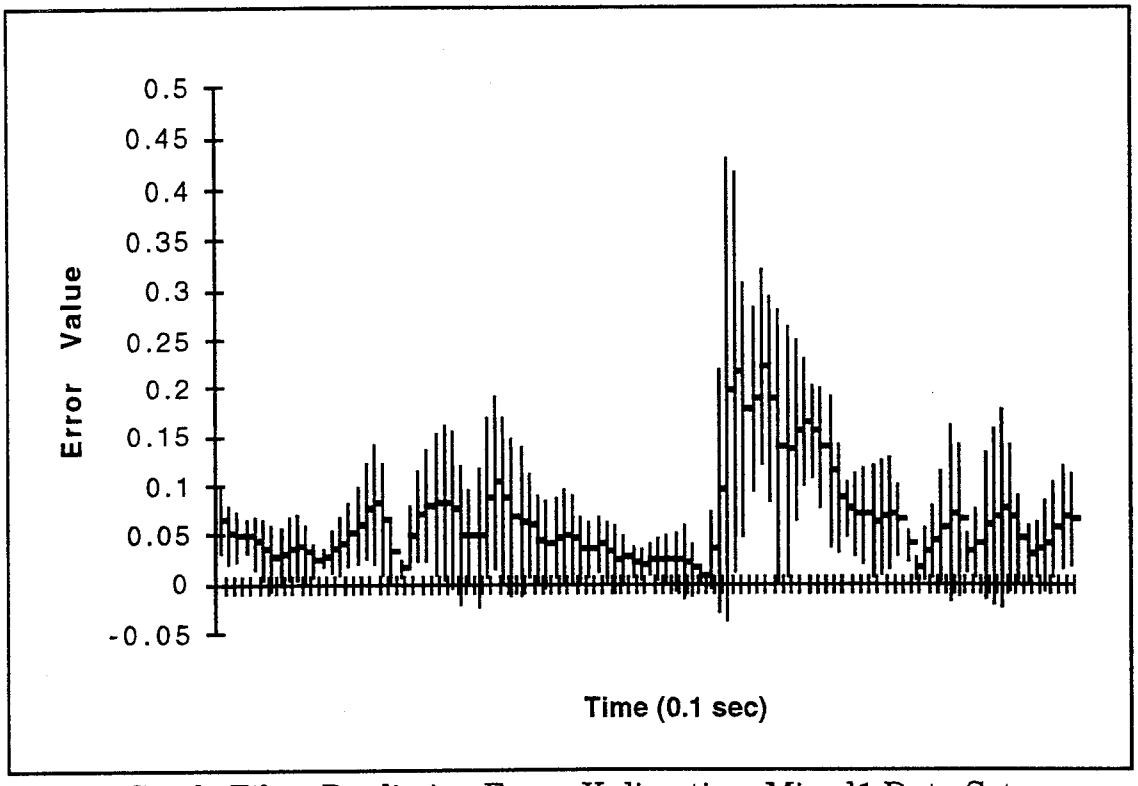
Z-direction Path Values, Mixed1 Data Set



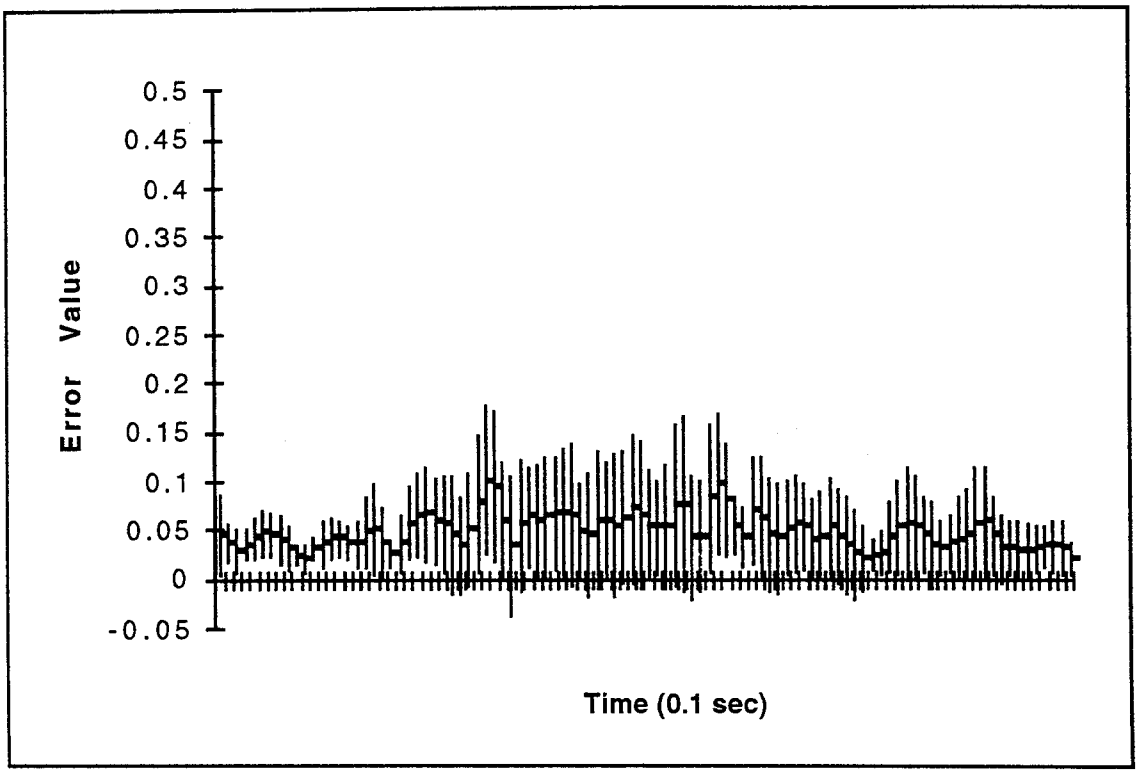
MMAE Prediction Error, X-direction, Mixed1 Data Set



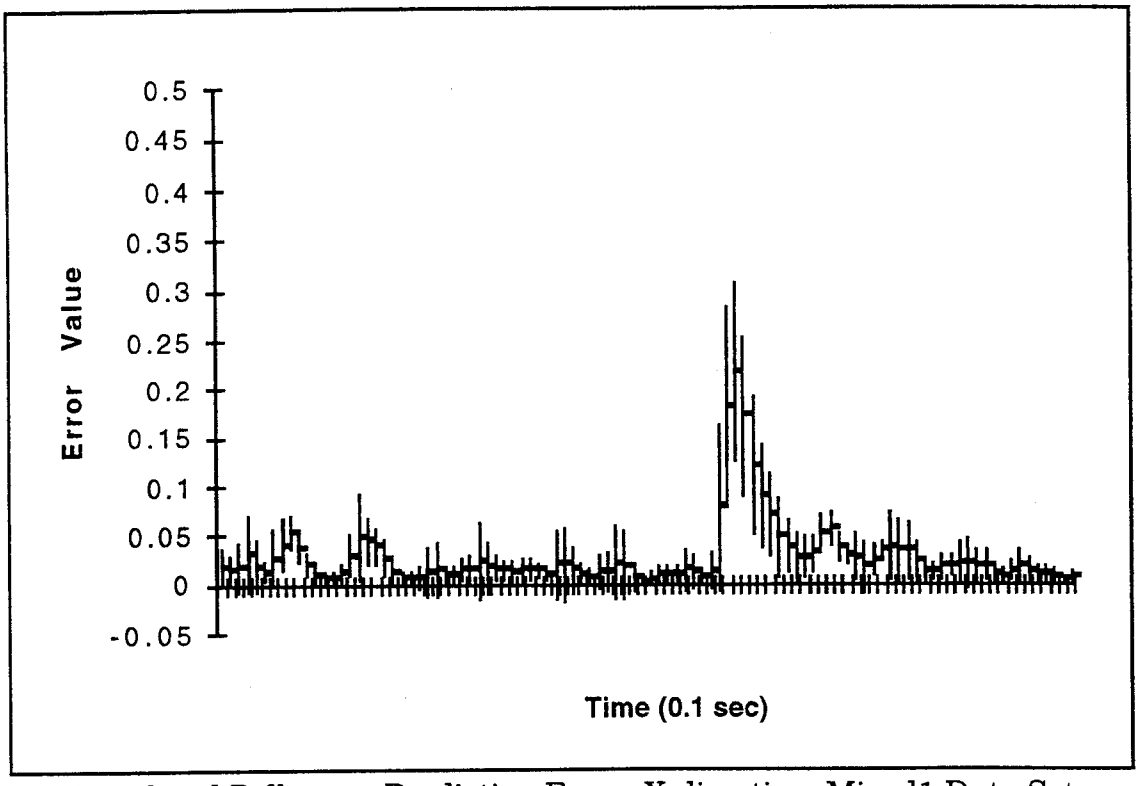
MMAE Prediction Error, Z-direction, Mixed1 Data Set



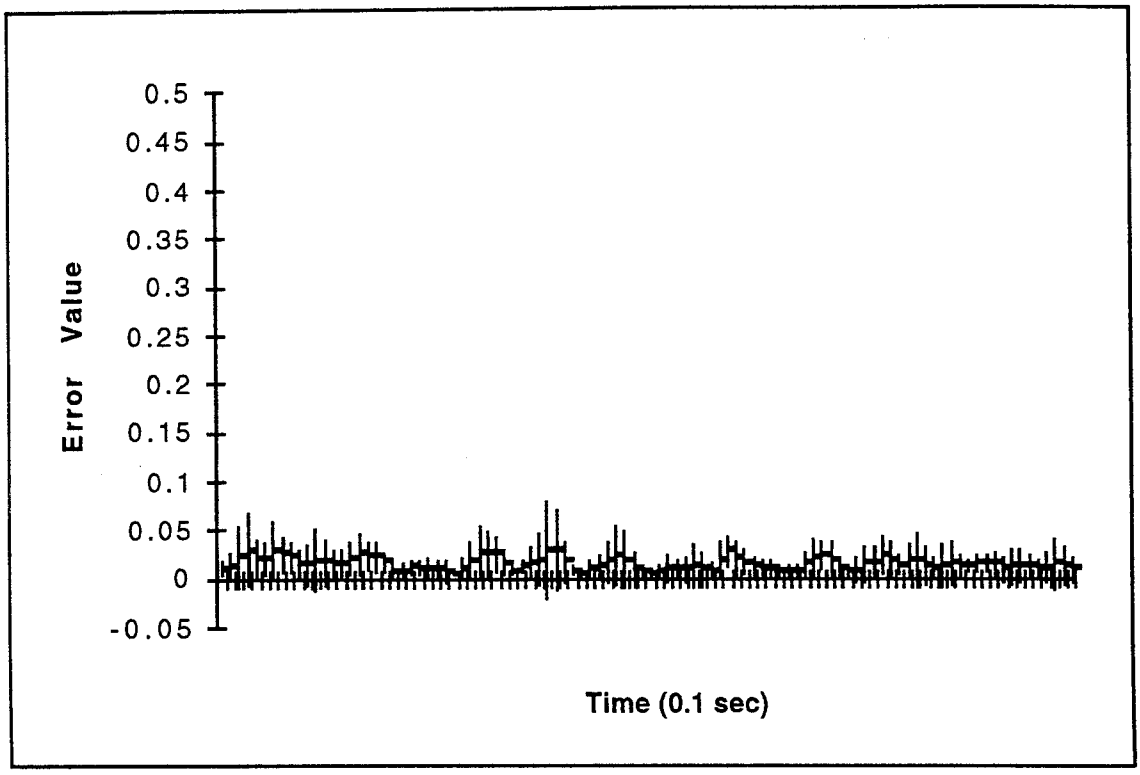
Single Filter Prediction Error, X-direction, Mixed1 Data Set



Single Filter Prediction Error, Z-direction ,Mixed1 Data Set



Simulated Polhemus Prediction Error, X-direction, Mixed1 Data Set



Simulated Polhemus Prediction Error, Z-direction, Mixed1 Data Set

Bibliography

[Arthur93] Kevin W. Arthur, Kellogg S. Booth, and Colin Ware; "Evaluating 3D Task Performance for Fish Tank Virtual Worlds"; *ACM Transactions on Information Systems*; Volume 11, issue 3, July 1993; pp 239-265.

[Azuma94] Ronald Azuma and Gary Bishop; "Improving Static and Dynamic Registration in an Optical See-through HMD"; *Proceedings of SIGGRAPH '94* (CD-ROM version).

[Batory91] Don Batory and Sean O'Malley; *The Design and Implementation of Hierarchical Software Systems Using Reusable Components*; Technical Report TR-91-22; Department of Computer Science, University of Texas, Austin; June 1991.

[Bryson92] Steve Bryson; "Measurement and Calibration of Static Distortion of Position Data from 3D Trackers"; *Proceedings of the SPIE – The International Society for Optical Engineering*; Volume 1669, 1992; pp 244-255.

[Bryson90] Steve Bryson and Scott S. Fisher; "Defining, Modeling, and Measuring System Lag in Virtual Environments"; *Proceedings of the SPIE – The International Society for Optical Engineering*; Volume 1256, 1990; pp 98-109.

[Chang84] Chaw-Bing Chang and John A. Tabaczynski; "Application of State Estimation to Target Tracking"; *IEEE Transactions on Automatic Control*; Volume AC-29, Number 2, February 1984; pp 98-109.

[Chidamber91] Shyam R. Chidamber and Chris F. Kemerer; "Towards a Metrics Suite for Object Oriented Design"; *Proceedings of OOPSLA '91*; pp 197-211.

[Chung92] James Chung; "A Comparison of Head-Tracked and Non-Head-Tracked Steering Modes in the Targeting of Radiotherapy Treatment Beams"; *Proceedings of the 1992 ACM Symposium on Interactive 3D Graphics*; 1992; pp 193-196.

[Diaz94] Milton E. Diaz; *The Photo Realistic AFIT Virtual Cockpit*; Masters Thesis, AFIT/GCS/ENG/94D-02, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; December, 1994.

[Ellis94] Stephen Ellis; "What are Virtual Environments?"; *IEEE Computer Graphics and Applications*; Volume 14, Number 1, January 1994; pp 17-22.

[Ellis91] S.R. Ellis; "Nature and Origins of Virtual Environments: A Bibliographical Essay"; *Computing Systems in Engineering*; Volume 2, Number 4, 1991; pp 321-347.

[Foley90] James Foley, Andries van Dam, Steven Feiner, John Hughes; *Computer Graphics, Principles and Practice* (second edition); Addison-Wesley Publishing Company; 1990.

[Fortner94] Jonathan L. Fortner; *Distributed Interactive Simulation Virtual Cassette Recorder: A Datalogger with Variable Speed Replay*; Masters Thesis, AFIT/GE/ENG/94D-10, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; December, 1994.

[Friedman92] Martin Friedman, Thad Starner, and Alex Pentland; "Device Synchronization Using an Optimal Linear Filter"; *Proceedings of the 1992 ACM Symposium on Interactive 3D Graphics*; 1992; pp 57-62.

[Gardner93] Michael Thurman Gardner; *A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions*; Masters Thesis, AFIT/GCS/ENG/93D-09, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1993.

[Garlan93] David Garlan and Mary Shaw; "An Introduction to Software Architecture"; in *Advances in Software Engineering and Knowledge Engineering*; Volume I; edited by V. Ambriola and G. Tortora; World Scientific Publishing Company; 1993; pp 1-39.

[Gerken91] Mark James Gerken; *An Event Driven State Based interface for Synthetic Environments*; Masters Thesis, AFIT/GCS/ENG/91D-07, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1991.

[Jagacinski77] Richard J. Jagacinski; "A Qualitative Look at Feedback Control Theory as a Style of Describing Behavior"; *Human Factors*, Volume 19, Number 4, 1977; pp 331-347.

[Kayloe94] Jordan R. Kayloe; *The Use of Ada 9X in a Visual Simulation System Software Architecture*; Masters Thesis, AFIT/GCS/ENG/94D-11, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1994.

[Kelley68] Charles R. Kelley; *Manual and Automatic Control*; John Wiley & Sons, Incorporated; 1968.

[Kestermann94] Jim Kestermann; *Immersing the User in a Virtual Environment: The AFIT Information Pod Design and Implementation*; Masters Thesis, AFIT/GCS/ENG/94D-13, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; December, 1994.

[Kozak93] J.J. Kozak, P.A. Hancock, E.J. Arthur, and S.T. Chrysler; "Transfer of Training from Virtual Reality"; *Ergonomics*; Volume 36, Number 7, July 1993; pp 777-784.

[Li94] X. Rong Li and Yaakov Bar-Shalom; "A Recursive Multiple Model Approach to Noise Identification"; *IEEE Transactions on Aerospace and Electronic Systems*; Volume 30, Number 3, July 1994; pp 671-684.

[Liang91] Jiandong Liang, Chris Shaw, and Mark Green; "On Temporal-Spatial Realism in the Virtual Reality Environment"; *Proceedings of the ACM Symposium on User InterfaceSoftware and Technology*; 1991; pp 19-25.

[MacKenzie93] I. Scott MacKenzie and Colin Ware; "Lag as a Determinant of Human Performance in Interactive Systems"; *Proceedings of the CHI '93 Conference on Human Factors in Computing Systems* (INTERCHI '93); 24-29 April 1993; pp 488-493.

[MatLab90] *PRO-MATLAB User's Guide*; The MathWorks, Incorporated, 21 Eliot Street, South Natick, MA 01760; January 31, 1990.

[Maybeck94a] Peter S. Maybeck; Class handouts, EENG 699, Special Study on Head Motion Tracking; School of Computer and Electrical Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; spring quarter, 1994.

[Maybeck94b] Peter S. Maybeck, Theodore D. Herrera, and Roger J. Evans; "Target Tracking Using Infrared Measurements and Laser Illumination"; *IEEE Transactions on Aerospace and Electronic Systems*; Volume 30, Number 3, July 1994; pp 758-768.

[Maybeck82] Peter S. Maybeck; *Stochastic Models, Estimation, and Control, Volume II*; Academic Press; 1982.

[Maybeck79] Peter S. Maybeck; *Stochastic Models, Estimation, and Control, Volume I*; Academic Press; 1979.

[McCabe89] Thomas J. McCabe and Charles W. Butler; "Design Complexity Measurement and Testing"; *Communications of the ACM*; Volume 32, Number 12, December 1989; pp 1415-1425.

[MultiGen94] *MultiGen Modeler's Guide*; version 14.0; Software Systems, 1884 The Alameda, San Jose CA; March 1994.

[O'Connor92] Bill O'Connor and Doug Blake; unpublished notes and software for a Multiple Model Adaptive Estimator developed at the Air Force Institute of Technology; dates on the software indicate it was developed in 1992.

[Oman90] Charles M. Oman; "Motion Sickness: A Synthesis and Evaluation of the Sensory Conflict Theory"; *Canadian Journal of Physiology and Pharmacology*; Volume 68, 1990; pp 294-303.

[Parr89] John M. Parr and Charles L. Philips; "State Estimation From Retarded Measurements"; *Proceedings – Energy and Information Technologies in the Southeast*; Volume 3 of 3, 1989; pp 1275-1280.

[Performer92] Patricia McLendon; *IRIS Performer Programming Guide*; Document Number 007-1680-010; Silicon Graphics, Incorporated; 1992.

[Polhemus90] *3Space Users Manual*; Document Number OPM3016-004D; Polhemus Incorporated, Colchester, Vermont; August 1990.

[PT01] *PT-01 User's Manual*; provided by O1 (a division of Optics 1, Incorporated); 3050 Hillcrest Drive Suite 100, Westlake Village, CA 91362.

[Rebo88] Robert Keith Rebo; *A Helmet Mounted Virtual Environment Display System*; Masters Thesis, AFIT/GCS/ENG/88D-17, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1988.

[Rheingold91] Howard Rheingold; *Virtual Reality*; New York: Summit Books; 1991.

[Rohrer94] J. J. Rohrer; *Design and Implementation of Tools to Increase User Control and Knowledge Elicitation in a Virtual Battlespace*; Masters Thesis, AFIT/GCS/ENG/94D-20, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; December, 1994.

[Rumbaugh91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen; *Object-Oriented Modeling and Design*; Prentice-Hall, Incorporated; 1991.

[Shaw93] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun; "Decoupled Simulation in Virtual Reality with the MR Toolkit"; *ACM Transactions on Information Systems*; Volume 11, Number 3, July 1993; pp 287-317.

[Snyder93] Mark I. Snyder; *ObjectSim – A Reusable Object Oriented DIS Visual Simulation*; Masters Thesis, AFIT/GCS/ENG/93D-20, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1993.

[So93] R.H.Y. So and M.J. Griffin; *Effect of Lags on Human Performance with Head-Coupled Simulators*; Technical Report AL/CF-TR-1993-0101; Human Factors Research Unit, Institute of Sound and Vibration Research, University of Southampton, Southampton, United Kingdom; June 1993.

[Stroustrup91] Bjarne Stroustrup; *The C++ Programming Language*, second edition; Addison-Wesley Publishing Company; 1991.

[Sutherland68] Ivan Sutherland; "A Head-Mounted Three-Dimensional Display"; *Proceedings of the Fall Joint Computer Conference*; 1968; pp 757-764.

[Tobin86] David M. Tobin; *A Multiple Model Adaptive Tracking Algorithm for a High Energy Laser Weapon System*; Masters Thesis, AFIT/GE/ENG/86D-37, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; December, 1986.

[USAF93] Department of the Air Force; *Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapons Systems, Command and Control Systems, Management Information Systems*; AFPAM 63-115; Washington: HQ USAF; November 1993.

[Vanderburgh94] John C. Vanderburgh; *Space Modeler: An Expanded, Distributed, Virtual Environment for Space Visualization*; Masters Thesis, AFIT/GCS/ENG/94D-23, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH; December, 1994.

Vita

Captain James E. Russell was born in Coudersport, Pennsylvania, on June 15, 1966. He graduated from Coudersport Area Jr. Sr. High School in 1984. He then attended the Pennsylvania State University (main campus), and in 1988 graduated with a Bachelor's Degree in Computer Science. Upon graduation, he received a reserve commission in the United States Air Force, and was assigned to Vance AFB, OK, for Undergraduate Pilot Training.

After nine months in pilot training, he was re-assigned to Air Force Global Weather Central (AFGWC), Offutt AFB, NE, as a communications-computer specialist. Before commencing his duties at AFGWC, however, he married his wife, Elaine, and attended the Basic Communications-Computer Officer Training course (BCOT) at Keesler AFB, MS.

Capt Russell started his tour at AFGWC as a maintenance analyst/programmer for the Data Transfer Interface (DTI) team. As a member of this team, he was responsible for maintenance of the DTI software that provides a standard interface to the Hyperchannel, which is the AFGWC intra-building communications network. Capt Russell eventually became chief of the DTI team, and after several months, was given responsibility for maintenance of Hyperchannel software as well.

After three years at Offutt, Capt Russell applied and was accepted to AFIT. His major was Computer Science with an emphasis in Software Engineering, and upon graduation from AFIT, he will be going to Kirtland AFB, NM, to work at Phillips Laboratory.

Permanent Address: 7 West Seventh Street
Coudersport, PA 16915

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
MULTIPLE MODEL ADAPTIVE ESTIMATION AND HEAD MOTION TRACKING IN A VIRTUAL ENVIRONMENT: AN ENGINEERING APPROACH

5. FUNDING NUMBERS

6. AUTHOR(S)
James E. Russell, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-21

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Mr. Donald J. Reifer
Chief, Ada Joint Program Office (AJPO)
701 South Courthouse Road
Arlington, VA 22204-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Software engineering tools and techniques were applied to design and implement an application that reduces lag typically present in virtual environment displays. The application was a Multiple Model Adaptive Estimator (MMAE), composed of three Kalman filters, that predicted head orientation one sample period into the future. The environment rendering software used these predictions to generate the environment display. Each of the filters in the MMAE was designed for a different assumed head motion type (benign, moderate, or heavy), which allowed the MMAE to adapt to changes in head movement characteristics.

The use of Ada 9X as an implementation language for a virtual environment applications was also investigated. Ada 9X provides object-oriented features for design and development, and it also offers software engineering support that makes it preferable to C or C++ for the application developed.

Two significant results were produced. The first is a performance baseline for the MMAE that can be used as a benchmark for future research in this area. The other is a performance-based comparison of equivalent Ada 9X and C++ graphics applications in which Ada 9X performance was practically identical to C++. This second result is somewhat surprising, and should be investigated further.

14. SUBJECT TERMS
Software Engineering, Kalman filters, MMAE, Virtual Environments

15. NUMBER OF PAGES
194

16. PRICE CODE

17. SECURITY CLASSIFICATION	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified GE	Unclassified T	UL