Sensory Interactions in Human Perception of Motion

by

Michael P. Markmiller

S.B. Mechanical Engineering
Massachusetts Institute of Technology, 1994

Submitted To The Department Of Mechanical Engineering In Partial Fulfillment Of The Requirements For The Degree Of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING
AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

Signature of Author: _____
tment of Mechanical Engineering
nber 6, 1996

Certified by: _____
Laurence R. Young
Apollo Program Professor of Astronautics
ıpervisor

Certified by: _____
Derek Rowell
chanical Engineering

Accepted by: _____
Ain Sonin
Graduate Officer, Mechanical Engineering

Sensory Interactions in Human Perception of Motion

by

Michael P. Markmiller

Submitted to the Department of Mechanical Engineering
on September 4, 1996 in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Mechanical Engineering

ABSTRACT

In most situations, we rely on a number of our senses to give us a useful understanding of our spatial relationship to the environment around us. A fundamental subset of this mental framework is the perception of our own self-motion. A large body of research exists on the ability of isolated senses (vision for example) to convey an impression of self-motion. The current paradigm is to take a systems approach to questions of how we perceive motion. This has worked well, and many input/output models are available for motion detection using single senses. However, single channel models do not reflect the complex interactions that take place when a person has all his senses to draw from. Recently, more emphasis has been placed on multi-sensory interactions.

In the current research, we chose to examine the interplay between visual, vestibular, and tactile cues in forming an individual's sense of self motion. We presented supine subjects with simultaneous Z-axis motion stimuli using a head-mounted display (HMD), the MIT Sled, which is a linear acceleration cart, and a G-seat, which is a tactile cueing device that acts by altering the distribution of pressure on the skin. The tactile pressure cues are similar to those present during whole-body acceleration. Motion disturbance profiles were relatively prime sums of sinusoids ranging from 0.0610 to 0.5005 Hz. The subjects attempted to maintain zero velocity by using a joystick controller whose output summed with the Sled disturbance.

We used a linear estimator model during analysis of the results. Subjects responses to vestibular inputs showed an approximate -20 dB / decade slope, which is consistent with an internal integration of accelerations as detected by the otolith. Tactile and visual gain curves were generally flat, with high degrees of variability. T-tests were performed on time response data to determine which cues subjects predominantly chose to null. These showed highly significant reliance on sled acceleration ($P<0.0005$ for most), and generally nonsignificant results for the other cues. Visual and tactile cues, when combined, were significant ($P< 0.005$). We conclude that use of the G-seat in a supine position will require reinterpretation of its cue polarity and possibly modification to the hardware.

Performance testing of the G-seat revealed it had a flat gain response curve out to approximately 10 Hz. Phase data indicated substantial lag above 1 Hz Integration of the three motion systems required several pieces of new hardware and computer code. These will serve as useful tools for further research into motion perception and sensory interaction.

Acknowledgments

This work was funded by NASA grant NAGW-3958, the number 4, and the letter Q.

This is the page I have been looking forward to writing for a LONG time. So many people have given of themselves to help me arrive at this point, and I am grateful to each.

Two years ago, I was a graduate-to-be sitting in Professor Young's office, wondering if I would be able to find a thesis project and advisor before going back to California for the summer. To that point, nothing else I had seen in the way of research could have held me for two years. Then Larry started talking about space physiology and building hardware, and I was hooked. And this project certainly has held me. And wouldn't let go. Still, this has been an excellent two years, both in experience, and in the friendships I have been blessed to have.

Larry, thank you for taking me under your wing and for the guidance you provided. It has been a pleasure to work with you (not for you, as you once corrected me). Best wishes to you, Jody and the family.

Although the homey decor was what first struck me upon joining the MVL, it was the people in it that made it special. Sherry, Karl, Keoki, Grant, and Jim were all a lot of fun to be around, and took time to lend a hand when it was needed. This past year, I've had the honor(?) of being one of the senior grad students. The new "crop" has been a kick to hang around with. It's with some sadness that I say goodbye to Adam, Dawn, Anna, Christine, Scott, Prashant, and Matt (whose singing I will not soon forget). Dave and Jen, I never would have guessed how absolutely off the wall both of you are. You saw me at my most frazzled and still could make me laugh. Jen, let us know when that Elvis hotel is finished. I'll rent a whole wing.

It's taken me two years, but I finally realize that this thesis is as much Patricia's as it is mine. Thanks for getting me to this point; I couldn't have done it without you. I hope you too have someone you can ask, "Now, tell me again why we're doing this?"

More than anyone else, it has been my wife, Jennifer, who has kept me going day by day. I can't express the joy it has been to have someone who loves me unconditionally. Why anyone would choose to marry an MIT grad student is beyond me (it must be the impeccable hygiene we all maintain). I stand amazed at the circuitous route by which He brought me to you. Here's to a lifetime of tacky card contests.

Finally, I give all the glory to the Lord, who knows my thoughts before they are mine. He has provided at every turn, and shown me mercy and grace. Two years of studying physiology has only deepened my conviction that He is the greatest designer, period.

"I praise you because I am fearfully and wonderfully made; your works are wonderful, I know that full well." Psalm 139:14

Table of Contents

(This page intentionally left blank)

(This page intentionally left blank)

Introduction

Each waking moment, we rely on our various senses to provide us with an awareness of our relationship to our environment. This flow of information is continuous and largely involuntary, yet somehow is precisely tailored to our needs at any given moment. Consider that, with only one pair of eyes, we can pick our way along a moonlit path, or weave our cars in and out of traffic on a busy freeway. Or, for those of us who have lived in California, how the slightest shaking of the ground under our feet can make us suddenly alert, wondering if this is the "Big One."

Our senses also work together, allowing us to accomplish amazing tasks. How does a gymnast or diver maintain such meticulous control over her body that she can launch herself into the air, bend, tumble, and twist violently, and land squarely on her feet, or barely disturb the pool? Somehow vision, balance, and touch come together seamlessly to provide the necessary information. Those of us who study the human body would do well to approach our work with a sense of wonder at the complex and brilliant design that each of us represents.

After nearly two centuries of rational, scientific investigation into human sensory performance (and after millennia of practical experience), we are only beginning to develop the tools and techniques for relating complex behavior to the underlying biology. As a result, research is most often focused on simple, well defined tasks. This has clearly been the case in the study of how we perceive self-motion. Whole body movement can be active, such as walking or running, or passive, as one experiences riding in a car or plane. Although all the senses may be involved, there are only a few, clear questions a person

must be able to answer : "Am I moving?" and "If so, where?" Without conscious effort, an enormous amount of information is condensed into a few key metrics. The awareness of one's motion is a fundamental sensory integration task.

The majority of the research in this field has involved studies carefully designed to isolate a single sense. Only after gaining an appreciation of the behavior of each sensory channel can we broaden the scope to include the effects of simultaneous stimuli. Our group, the MIT Man-Vehicle Lab, has taken part in many of the advances in our understanding of how vestibular and visual information contribute to one's sense of self-motion. Some of our findings, along with those of others, are described or referenced in this work. Those unfamiliar with sensory physiology may be surprised at how deeply the field has been influenced by engineering disciplines, and in particular, signal processing, system identification, and control theory.

Taking an engineering approach, the resulting paradigm is one of clearly defined inputs (acceleration, force, etc.) being acted upon by an unknown process or processes (the sensory organs, neural pathways, brain, expectations, etc.), and some form of measurable output. Choosing suitable inputs requires at least a basic knowledge of the physiology of the sensory organs. As for the processes to be identified (the "plant" in engineering parlance), a researcher will propose a model based partly on physiology, and partly on his expectations of how the biological system might act, given its usual function and the demands of its environment. Choosing a suitable output to measure can be difficult as well, because asking the subject to perform a task may influence his or her perceptions, clouding the picture.

A substantial amount of progress has been made towards characterizing both the physiology and the performance of individual sensory systems. Although such work will continue for the foreseeable future, the current trend is towards a more general understanding of how all the senses *combine* to form a single estimate of motion. This change introduces another layer of processing to the sensory model, where information from the various organs interact with higher level processes, such as expectations, or workload.

Mental set can have a significant effect on how an individual processes the information he receives from his senses. If he knows a certain motion is impossible, he will tend to discount motion cues that would indicate it, or will interpret the cues in a way that is more closely aligned with his expectations. This effect is not absolute, however - during experiments in vection (visually induced perception of motion) subjects often describe making motions that are clearly impossible (such as continually spinning without a concurrent change in position.)

Sensory conflict presents another wrinkle. In most situations, a person's senses provide mutually consistent information: visual, auditory, vestibular, etc. all more or less agree. When disparities arise, the brain is forced to discard information from one sense in favor of the others, or combine them into an averaged estimate of motion. Large disparities can produce motion sickness. Again, mindset plays a role in determining which senses to trust and which to discard.

The scope of this work

We propose a set of experiments in which human subjects are presented with simultaneous multi-modal motion cues. In particular, we are interested in the combination of vestibular, visual, and tactile information. Our intent is to obtain quantitative estimates of how strongly the subject relies on each of the three sensory channels. Interactions among the senses are of particular interest. For example, what overall self-motion does a subject perceive when his vestibular organs are not in agreement with the movement he infers from seeing the room pass by his eyes? Or will he assume he is accelerating when he feels himself sinking into his seat?

We have brought together three compete motion-cueing systems and combined them in order to concurrently stimulate all three senses. A linear acceleration cart, or "Sled", serves as a moving platform, into which a G-seat has been mounted. A G-seat is a chair whose seat and backrest are composed of arrays of inflatable bladders. By pumping air into or out of the bladders, the occupant can be presented with a "seat of the pants" tactile cue, much like sinking into a soft aircraft seat during takeoff. Additionally, subjects will receive visual motion cues using a Head Mounted Display (HMD). Through these three systems vestibular, tactile, and visual stimuli will be concurrently presented to the subject. He will then attempt to "null" his perceived motion using a joystick-type controller. By careful selection of the input disturbances, it will be possible to separate out the contribution of each modality to the subject's holistic sense of motion.

In the course of this research, quantitative data on the G-seat's performance will be gathered. This will be done to ascertain its utility in both this and future research.

Additionally, a substantial amount of hardware and software has been developed as was necessary for the experiments. It is hoped that these will be useful tools in later research.

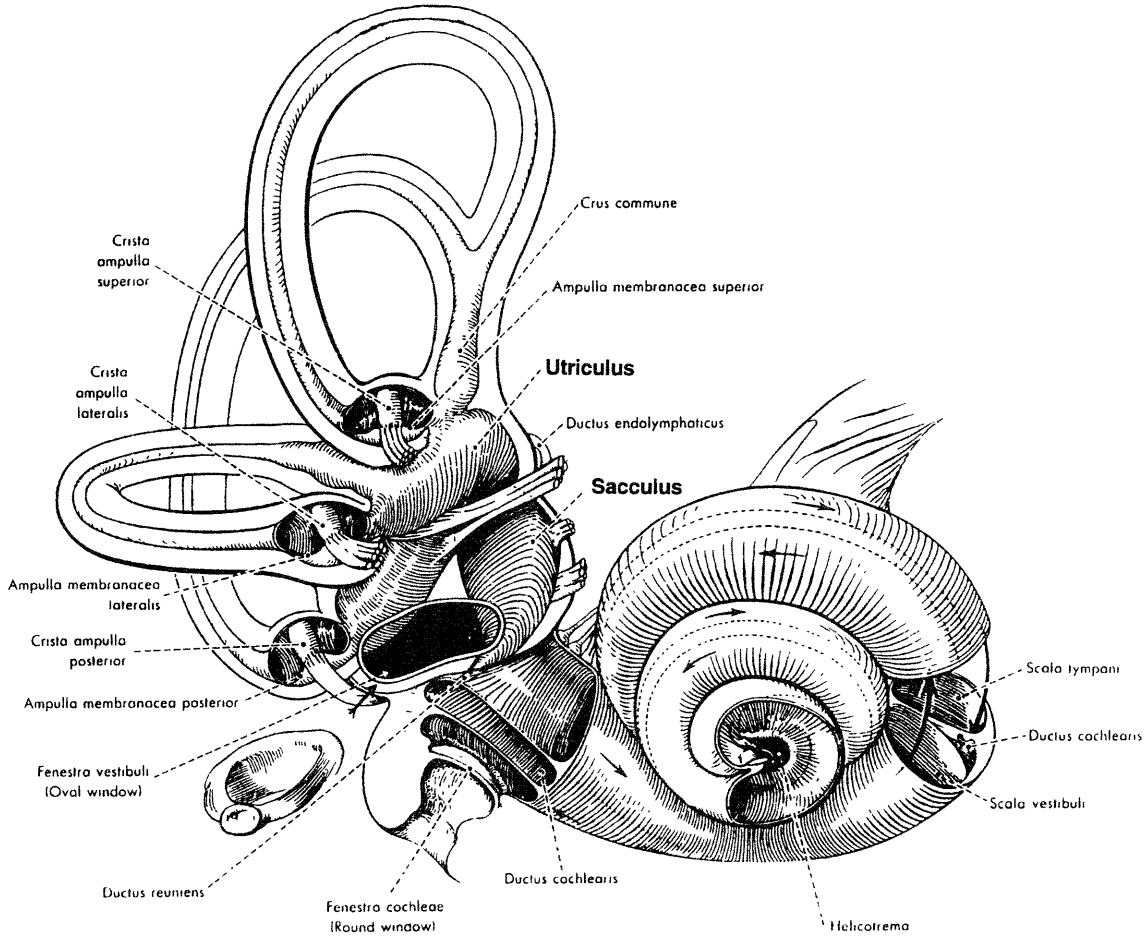


Figure 2-1 The inner ear. The vestibular system is to the left, with the semicircular canals particularly prominent. The utricule (utriculus) and saccule (sacculus) lay at the base of the canals. To the right is the cochlea. (Author unknown)

Vestibular (Otolith) System

Located in the inner ear, the vestibular system is the body's internal set of angular and linear accelerometers. As illustrated in Figure 2-1, a roughly orthogonal set of semicircular canals, stimulated by angular accelerations, and two seismic masses (the utricule and saccule), together relate head motions in six degree of freedom to the central nervous

system. Extensive research into vestibular function has shown its importance in balance, posture, and gaze stabilization.

Common to both the canals and the seismic *otolith organs* are hair cells. Their membranes are dotted with ion channels that open or close when the cells' cilia are strained. The ion flow generates a current that, if sufficiently strong, will lead to signal conduction along the nerve toward the vestibular nucleus in the brain. The cilia are coupled to an overlying membrane, which is impregnated with calcium carbonate crystals. This dense membrane, together with the flexible cilia and surrounding fluid (endolymph) acts as a mass-spring-damper system. Accelerations of the head (or changes in the gravito-inertial vector) lead to otolith motion, and resulting neural excitation. Fernandéz and Goldberg (1976) recorded the firing rates of individual neurons as a function of position, and found well defined relationships between tilt angle and firing rate. They described similar correspondence with linear accelerations.

Although afferent firing rates provide insight into vestibular function, models that span the gap between motion stimulus and resulting otolith motion estimate are more useful in the current research. One such model comes from Young and Meiry (1968). They sinusoidally accelerated human subjects, allowing the subjects to record their perception of motion using a joystick. By masking all but vestibular stimuli, they were able to create the model shown in Figure 2-2.

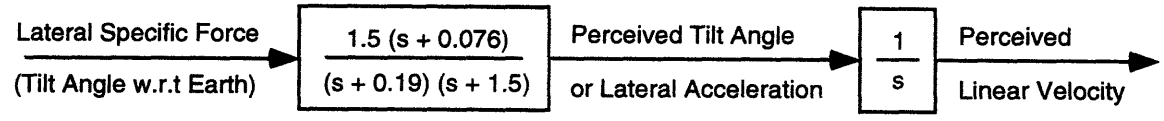


Figure 2-2 The Young and Meiry model. From Young and Meiry (1968)

Visual Stimuli - Vection

The existence of a visual component in the perception of motion has been long established. Early in this century, Helmholtz (1910) described the illusion of motion that can occur during optic flow, or movement of the visual field. This sensation is referred to as "vection." The best-known example of vection is the "train illusion:" a person in a stationary train sees another train passing by in one direction, and incorrectly infers that his train is actually moving in the opposite direction. One can only speculate why the terms "carriage vection" and "chariot vection" failed to catch on.

The train illusion is an example of *linearvection*, indicating translation. The other major class of vection is *circularvection*, involving perceptions of roll, pitch, or yaw. Both terms were coined by Fischer and Kornmüller (1931). In this research, we focus exclusively on linearvection.

Much of the research on vection has focused on which parts of the eye are responsible for it. Brandt, Dichgans, and Koenig (1972) proposed the peripheral dominance model. By obscuring various regions of a moving visual field, they determined that central vision was unimportant, and peripheral vision paramount, in generating vection. However, Warren and Kurtz (1992) list several experiments that contradict the peripheral dominance model, generating linearvection with as little as 7.5° of central vision. These findings bolster our decision to use the VPL Eyephone HMD, which has a field of view greater than 60'.

The Tactile Sense

The third, and final, sensory channel explored in this investigation is that of tactile, or haptic, cues. This refers to information passed to the central nervous system from pressure sensors in the skin. Acceleration of the body results in changes in the pressure distribution on the skin, leading to signal generation by these specialized cells. Following is a brief description of the anatomy and behavior of the major class of cells involved in this sense.

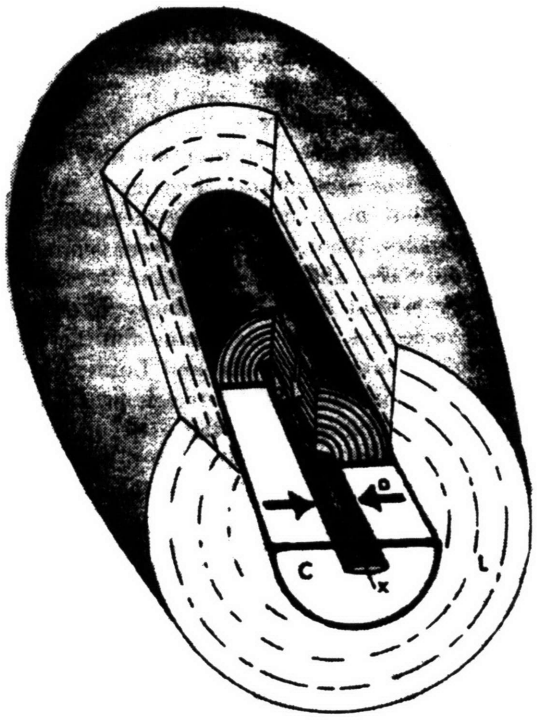


Figure 2-3 Schematic sectioned view of a Pacinian corpuscle. From Loewenstein (1971)

Pacinian Corpuscles

These large (1 x 0.6 mm) receptors were among the first to be investigated. In addition to their large size, they are present in a number of tissues, including skin, muscle, and within joints. (Lowenstein (1971)) Their most notable feature is an encapsulating sheath consisting of many layers, or lamellae. At the center of these lamellae lies the nerve, which

is responsible for the receptor's function. When the nerve ending is strained, ion gates in the cell wall open, producing an electric current as the charged particles move. This current is proportional to the strength of the mechanical stimulus - seemingly a complete mechanical transducer. However, this scaled response, or "generator current," is weak and would have difficulty traveling to the next synapse, often a distance of over a meter. To circumvent this problem, the generator current effectively undergoes an "analog to digital" conversion close to its source. The current triggers action potentials, which are all-or-nothing pulses of current that propagate along afferent nerve pathways. Stimulus magnitude is communicated by the rate of action pulse generation, akin to the firing of hair cells in the inner ear.

Mechanical strain in the surrounding tissue is the adequate stimulus for the Pacinian corpuscle (Lowenstein (1971)). The lamellae play an important role in that they act as a high pass filter for pressure signals. Horch et al (1977) found the Pacinian corpuscle responded only to mechanical acceleration, and not to displacement or velocity. Presumably, lower frequency signals tend to be absorbed in the layers, as they deform elastically and dissipate the energy. High frequencies produce large viscous forces, which propagate freely to the nerve core.

Several other classes of neurons contribute to the tactile sense. Two of these, the T_1, or Merkel, and the T_2, or Ruffini, cells respond to stimuli not detected by the Pacinian corpuscle. These slowly-adapting neurons detect displacement (Ruffini) or displacement and velocity (Merkel) in the surrounding dermis. Both classes show persistent responses to maintained displacements of the skin (Horch et al (1977)).

The tactile sense and perception of motion

The rapidly adapting nature of some skin receptors has important ramifications for the tactile sense. Steady-state signals, which are detected at their onset, quickly cease to be perceived. A common example of this phenomenon is the putting on and wearing of heavy clothing. After a short period of time, the wearer is no longer aware of the additional weight on his body. Clearly, adaptation has taken place, and it can be attributed, at least in part, to the receptor cells.

During whole-body acceleration (such as in a linear sled or aircraft), both the magnitude *and* distribution of pressure on the skin changes. That the magnitude changes is clear from Newton's third law, with force obtained from the integral of pressure over surface area. However, the distribution of pressure is affected by the compliance of the skin, underlying tissues, and of the apposed external surface (i.e. seat). For example, a military fighter pilot can experience more than 7 g's of upward acceleration ($+G_z$) during combat maneuvers. At these levels of acceleration, both the buttocks and the seat cushion have been substantially compressed. As a result, much of the force transmitted from seat to pilot passes through the ischial tuberosities, which are bony outcroppings at the base of the pelvis. The alterations in distribution contribute to the mechanical tissue strain, enhancing the sensation of acceleration over what would be seen were the pressure to change uniformly.

Multisensory Models

Multi-sensory models represent an effort to break through the obvious limitations of the single-channel models described in the previous sections. In all but the most artificial conditions, people draw from all their senses to build a picture of their orientation and motion. Therefore, in order to understand (and predict) perception of motion, it is essential to consider the information conveyed by all the senses.

It may not be enough, however, to simply piece together a collection of single-channel models and sum their outputs. Several factors contraindicate this approach:

1) Each sense organ has perception thresholds, noise, and saturation limits associated with it.
2) Response bandwidths are not identical.
3) Neural processing involves interactions that may emphasize or diminish the "importance" of a particular sense, depending on the state of the other senses.
4) Mental set affects the interpretation of sensory stimuli.

Each of the two models presented below addresses a subset of these concerns.

The "Internal Model"

As aircraft grow in complexity and cost, more and more pilot training is conducted in flight simulators. Therefore, fidelity to actual flight becomes paramount. Realism is, however, a vague metric, and a given simulator design cannot be truly evaluated until after it has been built and used. Mistakes can be expensive.

To evaluate a design before it is built, one can run its expected performance (in terms of accelerations, display specifications, etc.) through a model of the human pilot's senses, giving a prediction of motion as perceived by the pilot. This is the approach taken by the Air Force in the mid-1970's. The sensory interaction model that came out of this research is know as the "Internal Model," by Borah, Young, and Curry (1978).

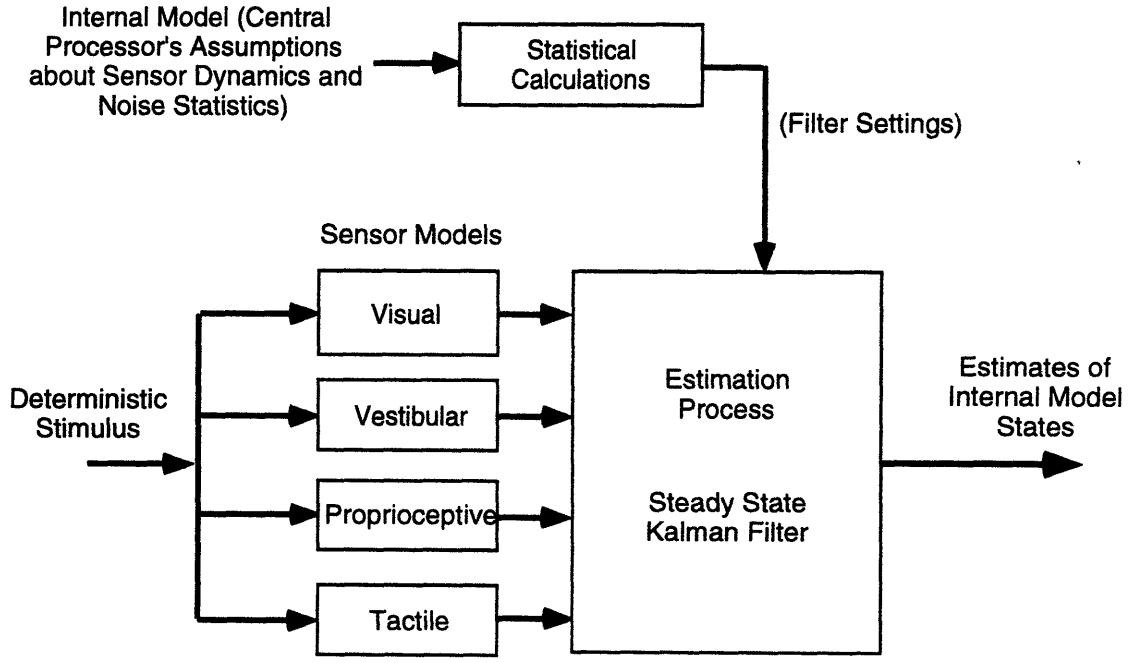


Figure 2-4 The Internal Model of human motion perception. From Borah, Young, and Curry (1978)

The overall structure of the Internal Model is shown in Figure 2-4. It consists of an optimal estimator (Kalman-Bucy filter) that accepts information from the various senses and combines them into an estimate of self-motion. The model derives its name from the manner in which the filter gains are set:

> "To assign optimal weights, the processor must have knowledge of the sensor dynamics, the expected stimulus spectrum, and the noise in each sensor

measurement. This set of assumptions is referred to as the Internal Model since it represents the system's knowledge about itself." (Borah et al (1978))

Notice that the individual sensor models upstream of the filter can be changed without affecting the overall structure of the model. This flexibility allows the substitution of better single-channel models as they become available. Similarly, internal models of various sensory interactions may be employed to find the optimal filter settings for particular combinations of stimulus modalities (e.g. visual-tactile, visual-vestibular).

Zacharias and Young Visual-Vestibular Conflict Model

Zacharias and Young investigated the sensory interaction that occurs during simultaneous whole-body and visual yaw. Uncorrelated sum-of-sines disturbances were applied to both a rotating chair and a projected visual scene. Subjects attempted to null their perceived motion using a control wheel; their commands were added to both chair and scene motions. Because the disturbances were uncorrelated, subjects were unable to simultaneously null both motions. Their responses were cross-correlated with the disturbances, and spectral analyses were performed. In this way, Zacharias and Young were able to discern the effects of both the vestibular and visual inputs on perceived motion.

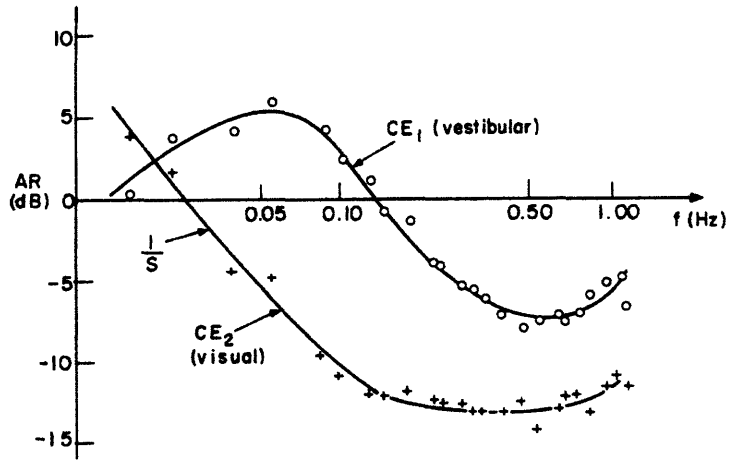


Figure 2-5 Control gains for visual and vestibular motion stimuli. From Zacharias (1977).

They proposed a dual-input describing function (DIDF) as an early model of the visual-vestibular interaction that they observed. Figure 2-5 shows the manual control gains associated with each input. Disagreement with single-channel results from other researchers led to the dismissal of DIDF in favor of the non-linear conflict model of Figure 2-6. The effect of the "soft switch" gain is to reduce the reliance on visual information when vestibular and visual cues disagree. In contrast, Berthoz et al. (1975) found that cue conflict led to dominance of *visual* over vestibular cues. The differences between angular and linear motion perception are acknowledged, however, and this comparison is offered only as a point of interest.

In the Zacharias and Young model, steady conflict signals are eventually washed out by the adaptation operator, allowing visual information back into the motion estimate. Further modifications to the model were made to correct discrepancies with experimental data for large amplitude velocity steps (>10°/s). It was later incorporated into the Internal Model described above.

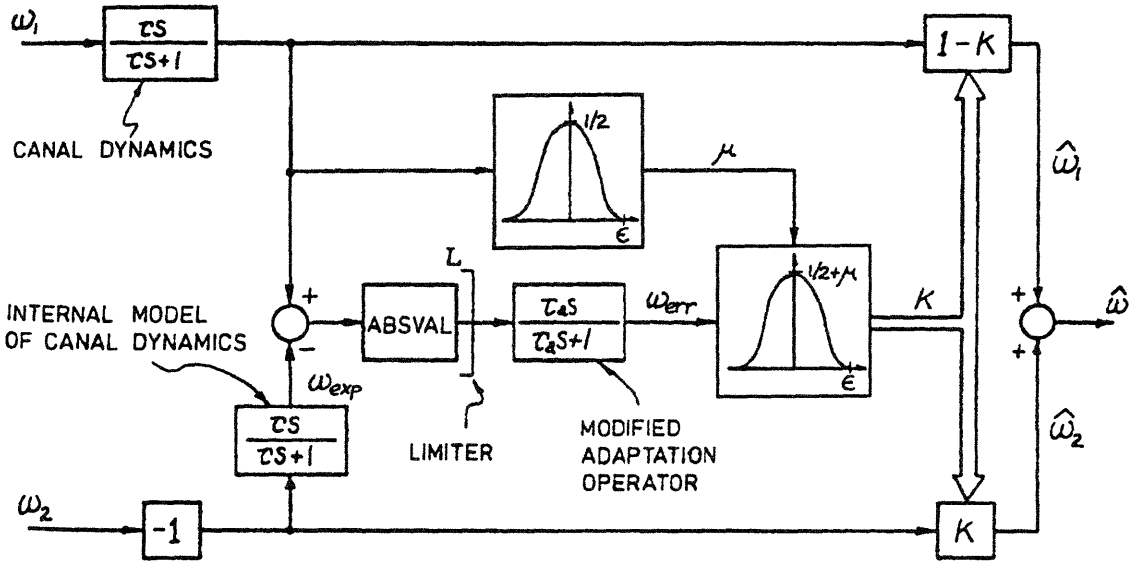


Figure 2-6 Zacharias and Young conflict model. From Zacharias (1977).

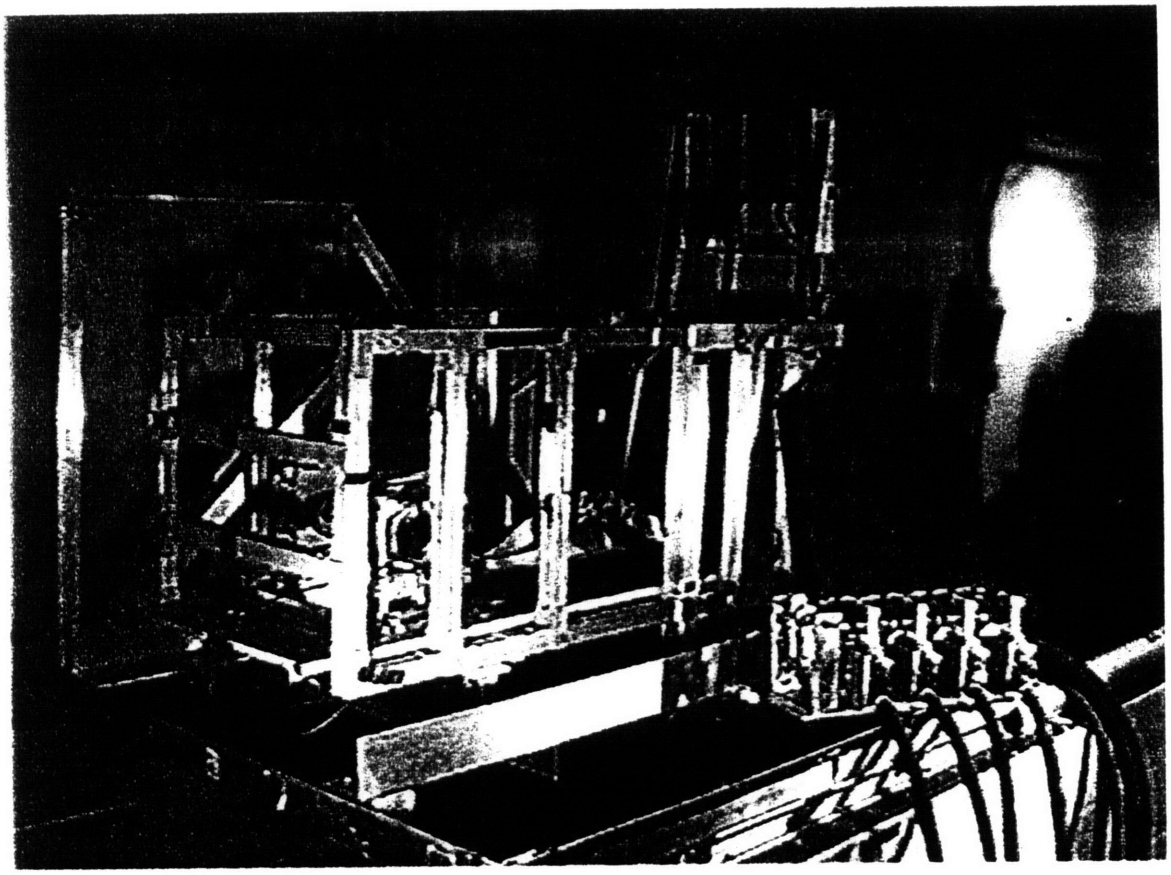


Figure 3-1 The NASA-Langley G-seat and the MIT Linear Sled. The G-seat back section is shown with its bank of air valves. The seat bottom has already been installed on the Sled and is obscured by the safety cage.

Equipment

The G-seat

A G-seat is an active device for stimulating the tactile sense through changes in pressure distribution on the skin. The system used in this research acts by inflating or deflating air bladders in the seat and back. The effect for the user is similar to the "seat of the pants" feeling used by fighter pilots to maintain control during air combat maneuvers. For the civilian, the most common situations to experience this sensation are on rollercoasters or during takeoff in a passenger jet.

G-seat History

The G-seat used in this experiment is an outgrowth of research into devices for improving flight simulator realism. The first descriptions of such a device date from 1958 (Johnson), and by the mid 1970's the US Air Force was using G-seats in research and in pilot training. These first generation seats employed an array of pneumatic bellows to change pressure distribution on the skin. Control was open-loop. These seats responded sluggishly, having a bandwidth of less than 1 Hz. (See Figure 3-4) As a result, they were suitable for sustained acceleration cues, but not onset cues.

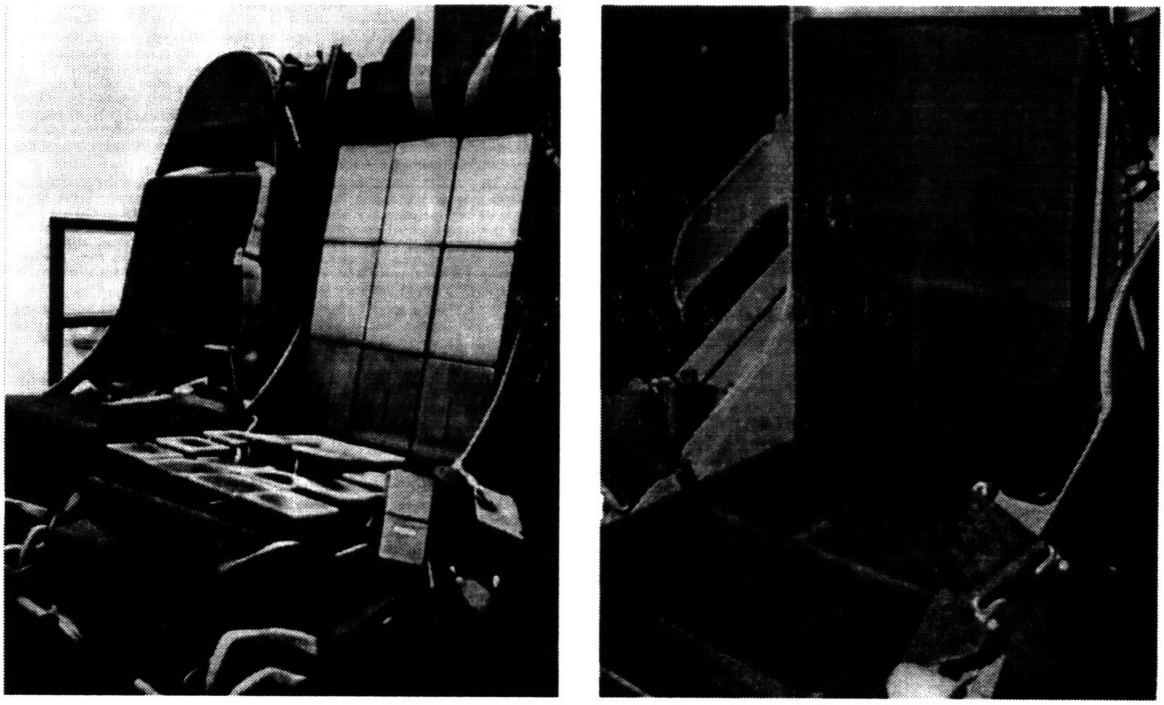


Figure 3-2 First generation (left) and NASA Langley (right) G-seats. (Left photo from Alberry and Hunter (1978). Right photo from Ashworth, McKissick, and Parrish 1984))

In 1980, the Advanced Low Cost G-Cueing System (ALCOGS) was developed by the Link Division of Singer Corp. for the Advanced Systems Division at Wright-Patterson AFB. Unlike the first generation seat, this package combined three simulation adjuncts: a G-seat, seat shaker (for vibration), and an anti-G suit, which is a garment of constricting

cuffs that prevent blood pooling during aggressive air combat maneuvers. Lower cost and improved performance were major goals. (Kleinwaks (1980))

The ALCOGS seat used hydraulic pistons for seat and backrest motion. A single pneumatic bladder in the backrest, and two in the seat, provided firmness changes. Closed-loop control (position and pressure) was used throughout, resulting in improved performance over the earlier G-seat.

The G-seat used in this research (Figure 3-1) can be considered a third-generation device, in that it departs from earlier versions by relying solely on rubber air bladders for all motion cues. NASA Langley Research Center designed and built several examples of this Seat for use in their motion simulators (Ashworth (1976)). NASA has loaned the MIT Man-Vehicle Lab one such seat for research into human sensory physiology under the auspices of NASA grant NAGW-3958.

Function of the seat

Unlike true whole-body acceleration (see Background - The Tactile Sense), a G-seat does not affect the total force exerted on the occupant (which is $= m\vec{g}$). Rather, it simulates acceleration by changing the pressure distribution on the skin and underlying tissue.

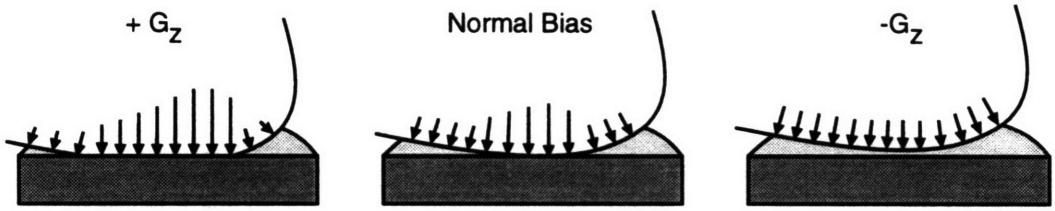


Figure 3-3 Changes in pressure distribution

In a properly adjusted G-seat, the subject is primarily supported on a cushion of air, with a portion of his weight transmitted to the hard seat surface under the bladders through direct contact. At rest, the subject quickly accepts (through adaptation, see Background) this particular level as "normal." To simulate upward acceleration ($+G_Z$), air is removed from the seat base, causing more of the subject's weight to be supported by the hard base, primarily below the ischial tuberosities (ITs). The subject interprets this new distribution as an increase in his own weight, consistent with positive G_Z. Furthermore, the subject sinks down into the seat a small amount (<2"), consistent with what occurs in real upward acceleration.

To mimic downward acceleration ($-G_Z$), air is added to the bladders, relieving pressure on the ITs. The subject rises a small distance "out" of the seat. Care must be taken not to overinflate the bladders - which would make them overly firm and reduce skin contact area, leading to false cues. A similar process occurs during simulation of forward-backward ($+/-G_x$) acceleration. Tilt and roll cues are possible through differential filling of adjacent bladders. A bias control allows adjustment for subjects of different weights.

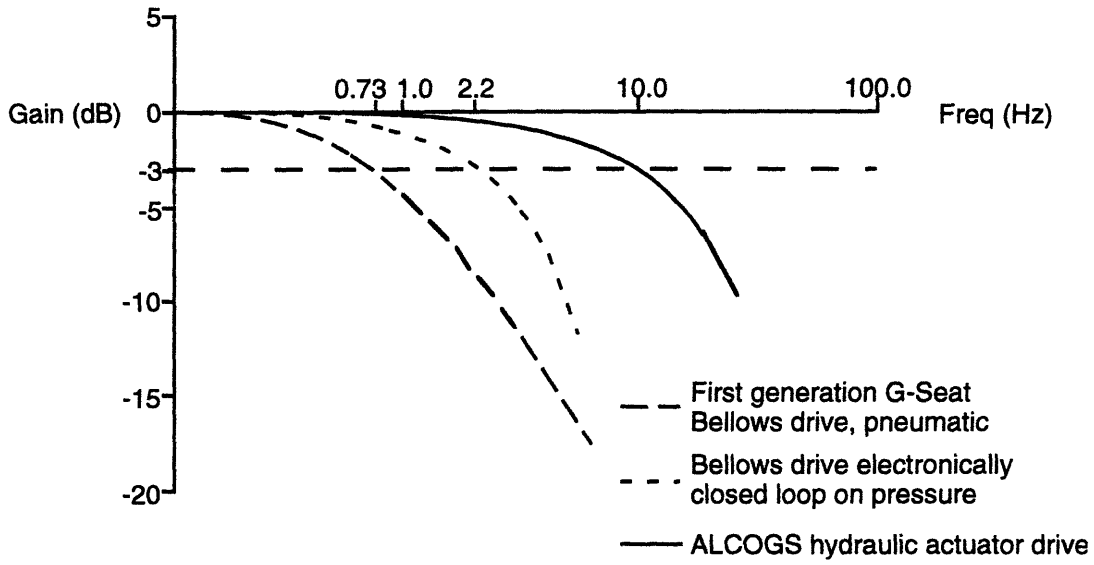


Figure 3-4 Performance of various G-cueing systems.

Previous research involving G-seat

The available literature on the G-seat deals solely with pilot performance in aircraft control tasks, such as tracking a target or maintaining a given heading. This is understandable, given the seat's genesis and history as a flight simulator component.

Ashworth et al (1984) found that the Langley G-seat (similar to the one used in this paper) produced a significant reduction in pilot lateral tracking error, both with and without simulator platform motion. However, the seat did not significantly lower vertical tracking error for this particular task. Others (Osgood et al (1988)) have evaluated the G-seat as an angular motion cueing device, and reported that a seat drive algorithm based on a combination of angular position and velocity was most effective (over position or velocity alone). For our research, we elected to use a drive model based solely on acceleration, rather than position or velocity, as Osgood et al chose. We believe that the underlying

dynamics involved in the linear case dictate an interpretation of the seat cues as accelerations.

Finally, McMillan et al (1990) tested a G-seat with both narrow and wide field of view (FOV) visual displays. In both cases, they found the seat improved pilot control over tests without seat motion. Their results suggest that wide-field visual motion does not take such precedence over tactile cues as to obliterate the effect of the latter. We hope that in our experiment, the subjects will be able to incorporate tactile information into their estimate of self-motion, despite conflicting wide FOV visual cues and concurrent vestibular stimulation from the HMD and Sled, respectively.

Detailed Hardware Description of G-seat System

A. Seat and Bladders

The bladders in both the seat base and backrest are arranged in 2 x 2 arrays. The front seat pads measure 8" W x 6.5" L x 1" D, and the rear 8" W x 8.5" L x 1" D at zero strain. Each bladder is composed of a poured and cured elastomer. At 5 psi, the unloaded pads expand to 2" in height. A short stem protrudes out the bottom of each and connects to the supply hose. The bladders sit on a rigid wooden base, which has cutouts for the hoses, and fabric is stretched over the bladders for appearance and protection.

Just upstream from each bladder is a National Semiconductor LX1802GN pressure transducer. These transducers provide the feedback signal used in the closed-loop pressure control (see below). A short section of 3/4" ID rubber hose then follows, which connects to a 1/2" IPS Schedule 40 pipe nipple. The pipe, which protrudes from the seat housing,

forms the interface with the supply hoses. Amphenol twist-lock connectors are used at all signal wire junctions.

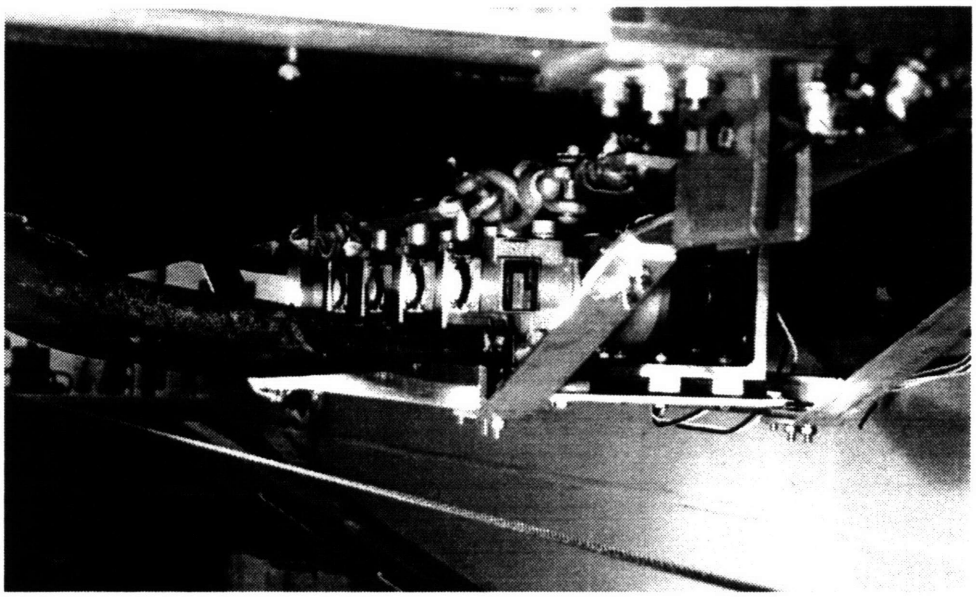


Figure 3-5 G-seat valves, shaker motors, and hoses mounted on the underside of the MIT Sled

B. Valves, Hoses, Air Supply

Air flow in and out of each bladder is controlled by a Alar Products MS24350-1 anti-G suit valve actuated by a Ling 203A Shaker. These are arranged in blocks of four, mounted on a 10" x 21" x 1/4" aluminum base. Each block has its own supply lines, electrical leads, low pressure regulator, vapor trap, and quick-disconnect fitting. Figure 3-5 shows one block mounted under the Sled.

The hoses connecting the valves to the seat bladders are 3/4" ID heavy rubber. We used a 380 ft^3 nitrogen gas cylinder, fitted with a two-stage regulator set at 70 psi. A 40' long, 1/4" ID "red rubber" gas hose ran from the cylinder to the valve block, along the sled umbilical. The low pressure regulator on the valve block was set to 10 psi.

C. Power and Servo Amps

Two separate electronics chassis serve the G-seat. The servo amps perform the pressure regulation. The inputs to this chassis are: external pressure commands (one for each bladder), a remote pressure bias and arming switch, and feedback signals from the pressure transducers. Trimpots allow adjustment of individual gains and biases. Outputs include pressure signals relayed from the transducers, and commands to the power amps.

The second chassis, containing the power amps, takes the commands (i.e. pressure errors) from the servo amps, amplifies them, and passes the signals to the shaker motors. We mounted the two chassis to the laboratory wall, and ran the cabling through the Sled umbilical.

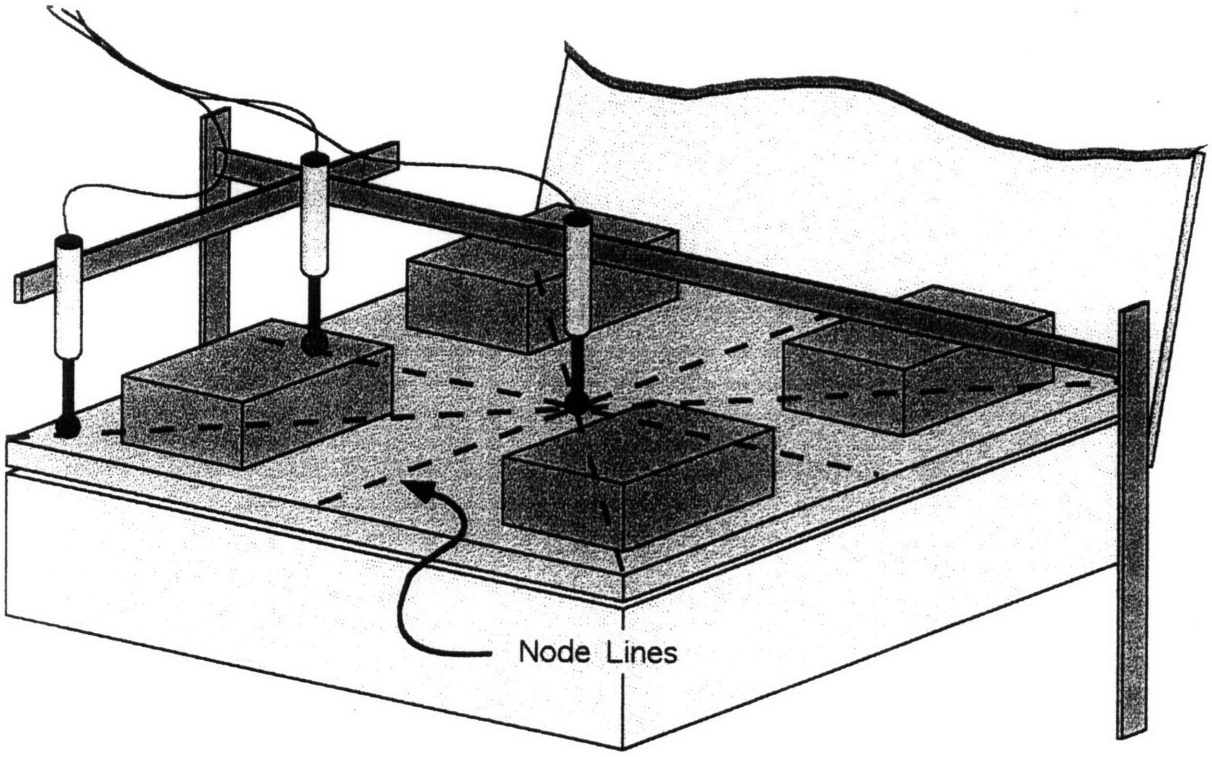


Figure 3-6 Testing rig used for assessing the G-seat's performance. The three cylinders represent linear potentiometers. A small U-joint at the tip of each connected to the flat plate.

D. Testing and Performance

Because our experiment depends upon the simultaneous presentation of various motion cues, and on analysis of the resulting output, it was of utmost importance to verify that the G-seat accurately produce the commands sent to it. NASA Langley had been using this model for both pilot training and performance experiments, and therefore their data on seat performance consisted of step responses and other criteria suited to flight simulation. We were interested in sinusoidal inputs; therefore we conducted a rigorous test of the G-seat's frequency response under a range of loads.

All testing took place with the seat in its full upright position. We commenced by placing a flat plate over the seat bladders. Unlike the ALCOGS, this G-seat has no provision for position feedback. As a result, it became necessary to add position sensing devices. The plate was instrumented with three linear potentiometers, as shown in Figure 3-6. By locating the pots at one corner, along one side, and in the center of the plate, we were able to discern vertical displacement, roll, and pitch. The signals from these pots were sent to the Pentium data computer (see Equipment - Sled), which also issued the pressure commands to the G-seat controller.

The first testing task involved adjusting the individual bladder gains to achieve a level vertical displacement of the flat plate, from a single pressure command sent to all the bladders. We did this by adjusting the gains until all three pots reported equal displacements. We then began a series of dynamic tests over a range of frequencies and loads. The frequency range was 0.095 - 16.1 Hz, and the loads varied from 30.5 to 229 lb., achieved with combinations of lead bricks distributed over the flat plate. At each set of parameters, we recorded a number of cycles, and calculated gain and phase for each using MATLAB. A subset of these results are given in Figure 3-7. The bandwidth of 10 Hz compares well with that of other G-seats. Conversion from height to pressure is possible using a non-linear transformation. However, this would not alter the shape of the frequency response curves.

We set out to determine this particular G-seat's suitability as a research tool. The dynamic performance tests described above showed the G-seat will function reliably throughout the frequency range (≤ 0.5 Hz) we intend to employ. We performed no quantitative tests as to

the realism of the G-seat tactile cue, choosing to rely on qualitative impressions of the seat in the upright position (which were acceptable) and reports of previous G-seat use.

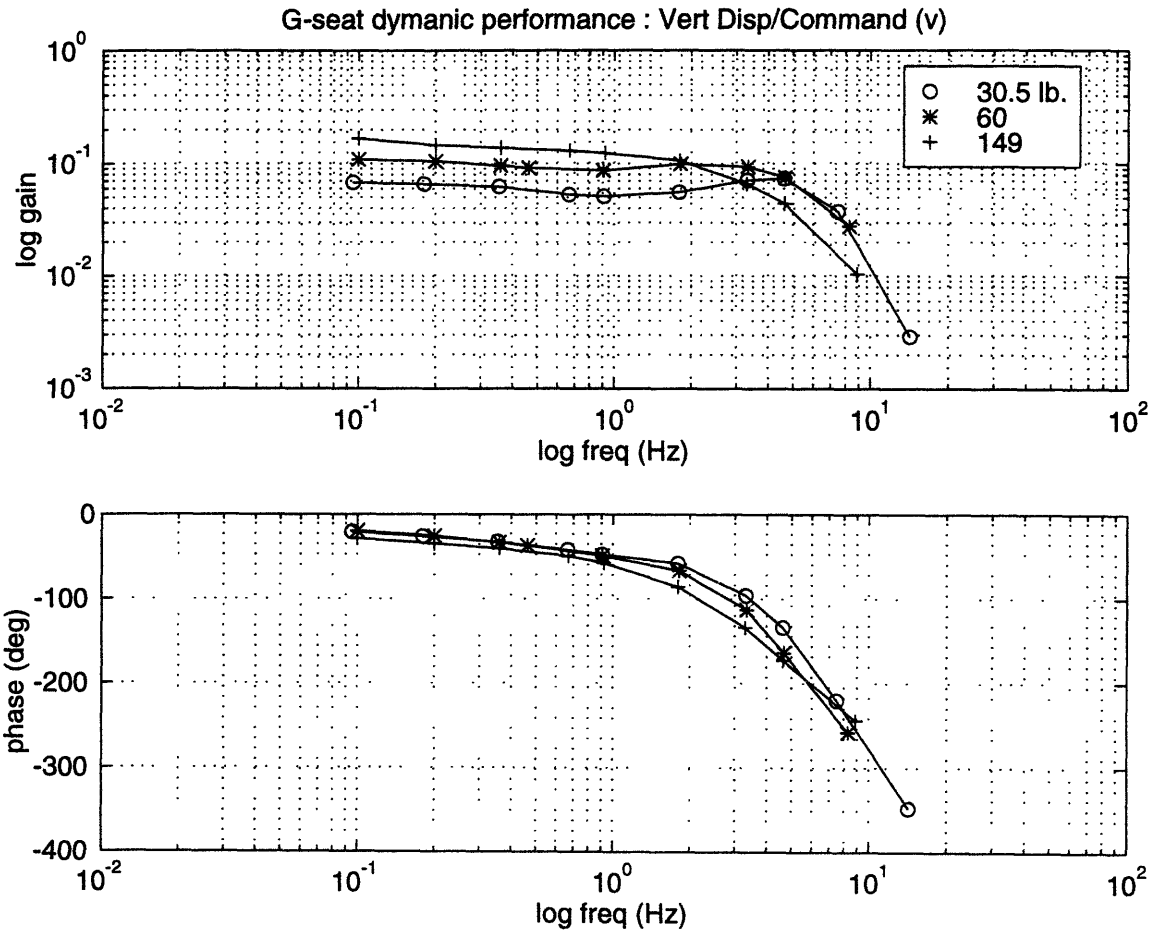


Figure 3-7 Bode plot of G-seat performance. The gain represents vertical displacement to commanded pressure for a range of loads.

The Head Mounted Display

In light of the rapid advances in "virtual reality" tools and displays, we have elected to provide all visual cues used in this experiment using computer-generated moving scenes. Several considerations, which are discussed below, led to this choice.

Our visual display system centered around a VPL Eyephone 1[1], which is a stereoscopic Head Mounted Display (HMD), capable of displaying 360 x 240 pixels, and subtending a visual arc of 80° horizontally and 60° vertically in each eye. It has a maximum refresh rate of 60 frames per second. A system of straps and a lead shot counterweight hold the Eyephone against the face and reduce user fatigue.

The Eyephone has been used in other MIT Man-Vehicle Laboratory research (DeSouza (1995), Skwersky (1996)) to successfully generate circularvection, which is the illusion of yawing, rolling, or pitching in a fixed visual environment. This sensation usually arises after several seconds of exposure to a uniformly rotating visual field that extends into the visual periphery (Brandt et al (1972)). Although the Eyephone has a limited field of view, its success at producing circularvection contributed to its selection in our experiment. We expect it will provide a stimulus compelling enough to generate linearvection, the translatory analog to circularvection. This phenomenon is more fully discussed in the Background section.

A Packard Bell 100 MHz Pentium computer running World Tool Kit[2] generates the visual environment. Twin Spea I860 FIRE graphics accelerator boards digitize stereo video streams and perform real-time conversion of the video signals from digital to NTSC standard.

World Tool Kit is a C based suite of programs and procedures that provides tools for creating and manipulating 3D graphical objects, navigating around virtual environments (VE), and support for various input devices, such as head trackers. By incorporating

[1] VPL Research, Inc. Redwood City, CA

WTK functions in his application, a programmer is free to focus on the more creative tasks involved in making a VE. Because they are C based, programs written with WTK tend to be fast and flexible.

The WTK application developed by John DeSouza and Adam Skwersky for Dr. Charles Oman's NASA Neurolab experiments in circularvection served as the platform for our visual scene. As their program supported only rotating scenes, extra code was written to perform the linear translations used in our experiment. The source code, including new code in bold, is given in Appendix A. Briefly stated, the executable program is composed of several modules: one that loads in motion profiles, another to perform these scene motions in a scene, and a third to handle the user interface, which is a series of menus for the operator. The experiment profiles, and scene and motion descriptions, are contained in text files and are accessed at runtime. The end result of this program architecture is that changes to the scenes and motions are very easy to make, giving this system a distinct advantage over mechano-optical devices, which tend to be very difficult to modify once built.

A particularly insidious problem we encountered using WTK on the Pentium was a total lack of support for regulating display frame rates. Such a system, allowing no external speed control, is known as "free-running." The practical result is that scene motion speeds up and slows down depending on the complexity (number of polygons) of the scene. Skwersky (1996) reported that these apparent accelerations of the visual field, caused by fluctuations in the frame rate, caused alteration or loss of the vection sensation. His solution was to "texturally balance" the rotating room by maintaining scene complexity at a

---

[2] Sense8 Corp. Sausalito, CA

constant level throughout the trial, which diminished rate fluctuations. This approach was not possible in our experiment, because linear motion tends to bring objects into and out of the field of view.

Therefore, several attempts were made to regulate the program frame rate. The first attempt involved using pulses from the computer system clock to time frames. All attempts at this approach failed, however, because on the PC, the clock is updated only 18.2 times a second, even though resolution of 1/100 s is claimed. We found that it is possible to increase the clock rate using assembly language instructions, but this was beyond the scope of the author's programming ability.

The solution we found was as follows: an external function generator (Krohn-Hite Model 5100A) was connected to the Clear to Send (CTS) and Signal Ground (SG) pins on the computer serial port (COM2). During simulation, a C function waits for a rising edge on the CTS port before allowing a frame to be rendered. In this way, the function generator governs the simulation speed. Care must be taken to set the frequency below the minimum free-running frame rate, or missed cycles may occur. Furthermore, the Ring Indicator (RI) pin was connected to the trigger channel on the Sled console, allowing synchronized starts of the Sled, G-seat, and visual scene.

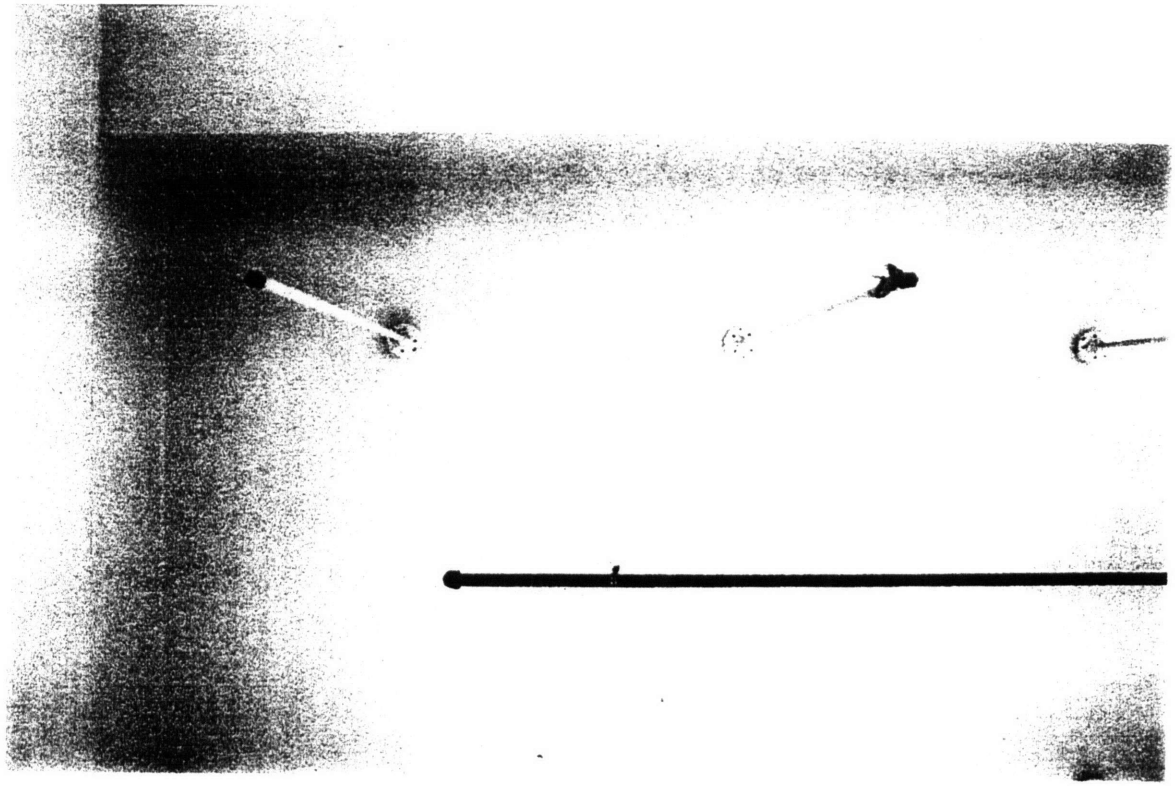


Figure 3-8a Subject's view of the laboratory while laying supine in the Sled.

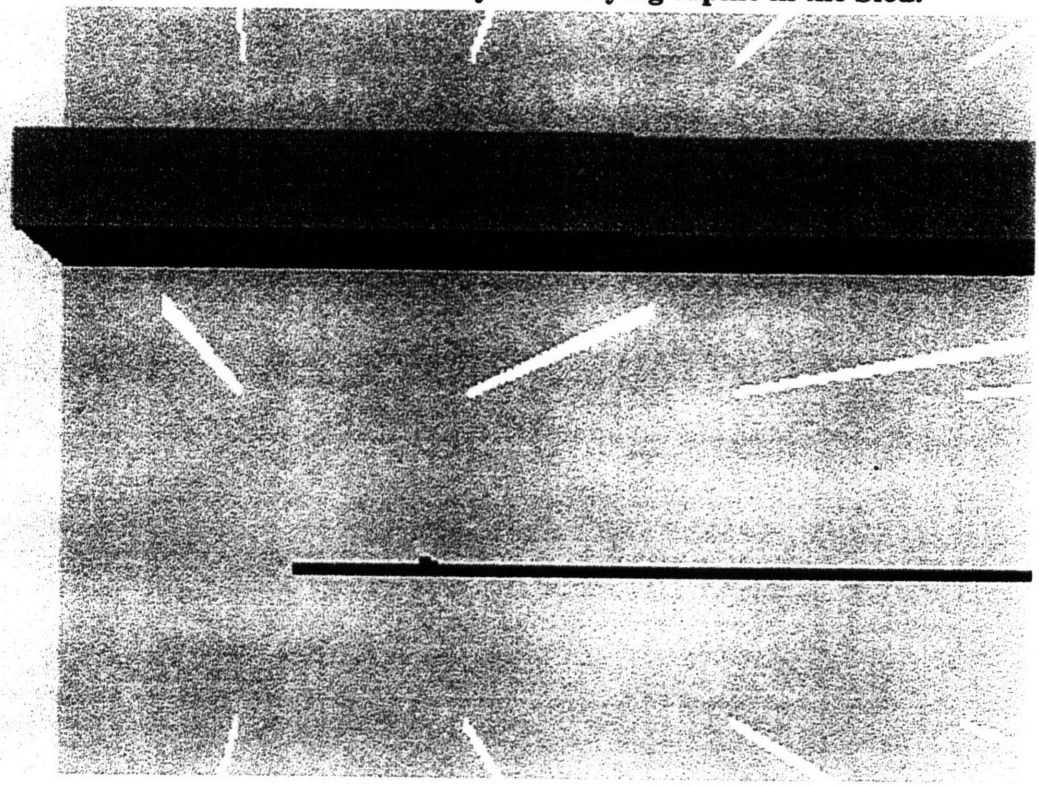


Figure 3-8b Subject's view of the virtual room. This is a monoscopic representation of the stereoscopic images presented through the HMD.

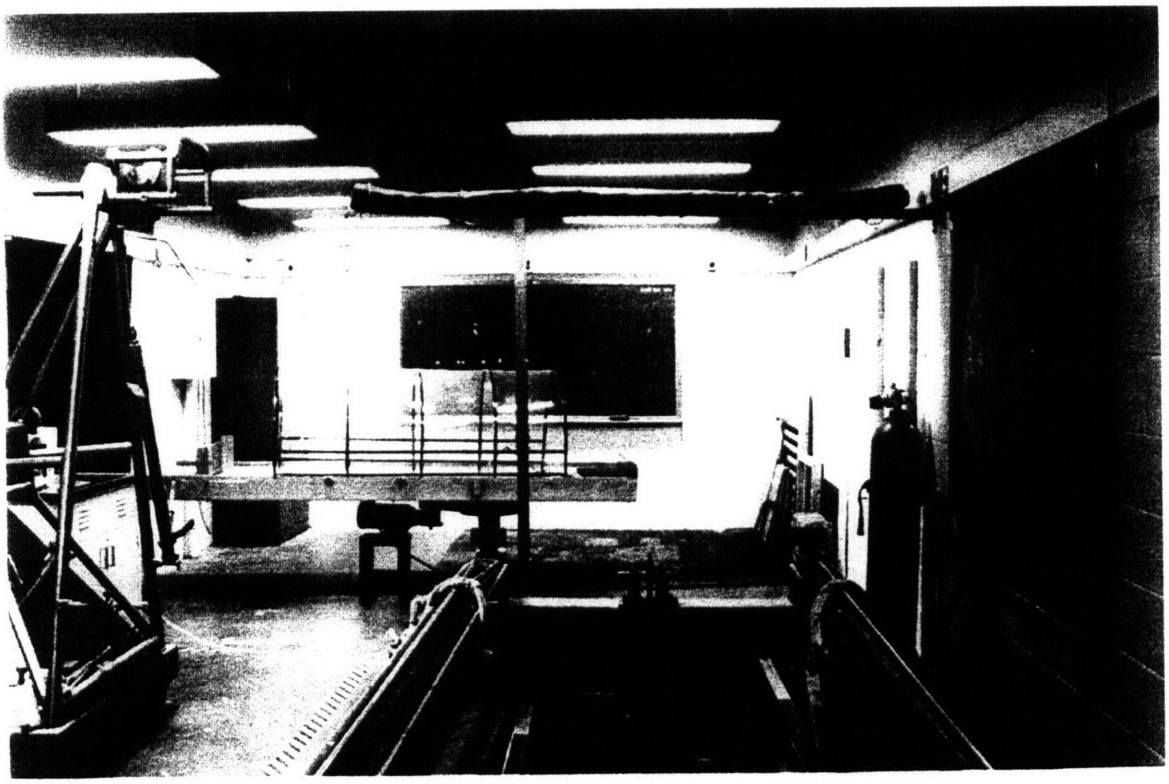


Figure 3-9a Subject's view of the laboratory while sitting upright in the Sled.

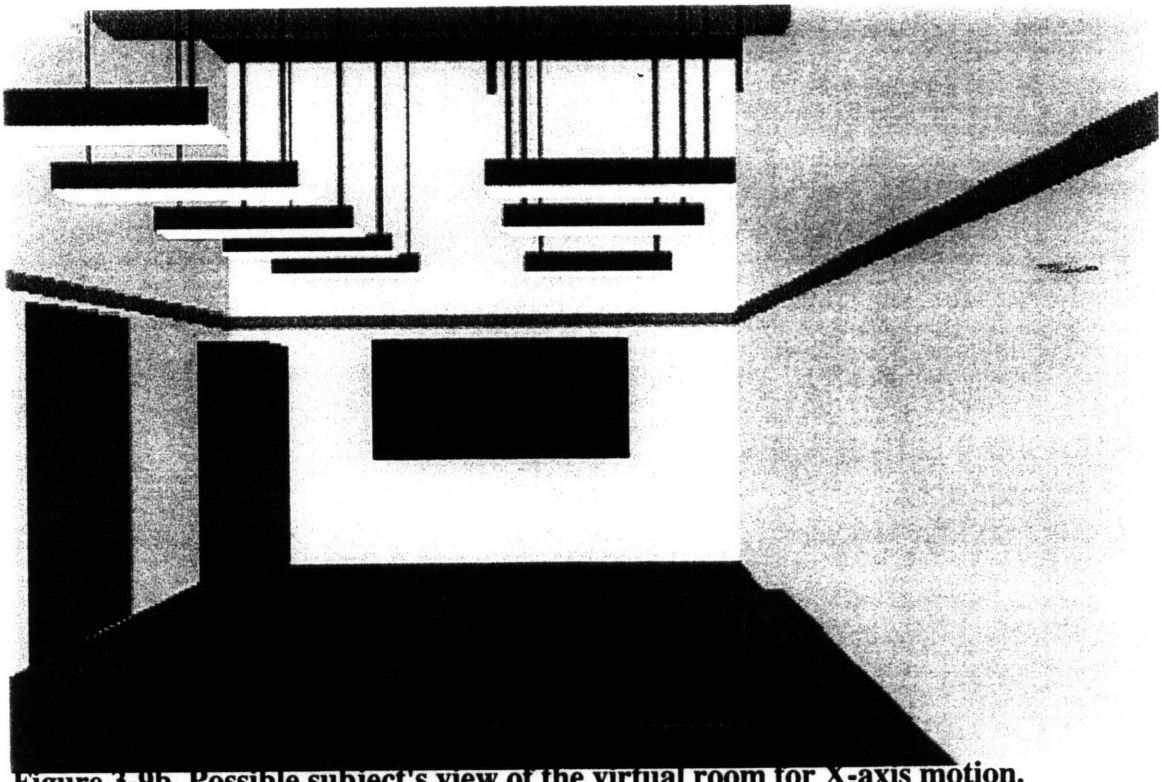


Figure 3-9b Possible subject's view of the virtual room for X-axis motion.

The visual scene used in this research is shown in Figure 3-8b. The "room" is a scaled copy of the actual laboratory in which the experiments were conducted (Figure 3-8a). Differences in perspective between the rendered image and the photo above are due to limitations on the film camera's optics, and not the HMD. The perspective of the computer generated scene matches the unaided view much more closely. To create the scene, we first measured the room and its contents, paying particular attention to those features that would be most recognizable, such as the light fixtures (during the trials, subjects are supine, and see only the ceiling, lights, and upper walls.) These dimensions were then used to create a 3D model using 3D Studio, a modeling and rendering program by AutoDesk. Another advantage of this computer-based approach is that future research along other subject axes (e.g. dorsal-ventral) can make use of the same scenes and equipment. Figures 3-9a and 3-9b show its applicability for looming (±X-axis) motions. Substantial effort went into removing unseen or unimportant features from the scene, resulting in simpler (and faster rendering) graphics. The 3D Studio files were then converted to ASCII Neutral File Format (.NFF) in WTK and used with the experiment software. These files are given in Appendix C.

The Sled

The MIT Sled was used to provide whole-body motion cues for this experiment. The Sled is a computer controlled, rail-mounted cart with seating for a single subject. The seat and enclosure can be reoriented to allow motion along the subject's major body axes: anterior-posterior (X), inter-aural (Y), and rostro-caudal (Z). Total track length between safety stops is 4.7 m.

Subject Enclosure

The Sled has gone through several major revisions since the first version used by Meiry in the mid-1960's (Meiry (1965)). The current version dates from 1979, which Lichtenberg (1979) constructed to perform experiments related to Spacelab-1. In its current form, the subject seat consists of a cushioned seat and fully-upright backrest. Safety harnesses are standard 5-point aircraft type. A rigid footpan with nylon restraints holds the subject's feet in place. The head restraint consists of two padded plates that are screw-adjustable. Bose headphones with active noise cancellation are contained within the head restraint, as is a microphone for communication with the experimenter.

The protective enclosure is largely of welded box section aluminum, with the segments around the subject's head composed of heavy fiberglass. This was to accommodate an early eye-tracking system that measured eye position magnetically using contact lenses with embedded wires. Readers interested in this system are encouraged to read Law's (1991) thesis. Wind and light cues to the subject are minimized by placing heavy black fabric over the protective frame.

Motor, Cable, and Controller

The Sled is propelled along the track by a loop of steel cable that is attached to the underside of the cart and wrapped around an idler and a drive pulley. The drive pulley is mounted on the shaft of an Inland 3.5 HP permanent magnet servo motor. The motor controller is a Pulse Width Modulation (PWM) type (General Electric HiAk) that uses a motor tachometer for velocity feedback. Additionally, a potentiometer on the motor shaft is used for positioning but not for velocity control. Hiltner (1983) states that the Sled has a

maximum acceleration of 10.0 m/s^2 and a bandwidth of 7 Hz. However, in the current research these limits will not be approached.

Sled Computer

Sled motion profiles are created and executed on a Northgate 386 PC running "Sled," a C program first used by Christie (1992). Velocity commands from the computer are passed through unity gain op amps and sent to the Sled controller. A variety of profiles are possible, including sinusoids, damped steps, and ramps specified in position, velocity, or acceleration. The program can also send commands through a separate "AUX" channel to drive another piece of equipment. In this manner we were able to independently command the G-seat. The sum-of-sines waveforms used in this experiment are described the Methods section.

Data Collection

Collection of data from experimental runs was done on a 90 MHz Pentium computer. A Keithley Metrabyte DAS-1601 board provided 16 single-ended analog input channels. Using bipolar ranges of ±10 V, the 12 bit A/D converter gave a resolution of 4.88 mV. A Keithley Metrabyte DDA-08 analog output board was also installed, although this board was used only during dynamic testing of the G-seat. We used Labtech Control software to perform the data acquisition. This is an icon-based data acquisition and control program, similar to National Instruments LabView in function and style. Data was sampled at 50 Hz. Appendix D contains an illustration of our data acquisition program.

The following signals were recorded during runs for later analysis: sled position, velocity, and acceleration; subject joystick position and sled velocity command; and G-seat bladder pressure transducer signals. Sled position and velocity are taken from the position potentiometer and tachometer on the motor shaft, respectively. Acceleration is determined by an accelerometer on the Sled cart. The programmed velocity profile is the sum-of-sines executed in the Sled program. During closed-loop runs, the Sled velocity command is the sum of the subject joystick signal and motion profile. The summing is performed by the Sled computer, and the velocity command is what is sent to the velocity controller.

Experimental Methods

The hardware and software described in the previous sections have not previously been used together in a research setting. Therefore, we developed a motion-perception experiment that could serve two major goals: first, to better understand how humans perceive self-motion through their various senses, and second, to assess the combined performance of the G-Seat, visual display (HMD), and Sled. In particular, we were interested in describing the interactions that occur between tactile, visual, and vestibular cues. Hopefully, the results will add to existing models of motion perception, such as the Optimal Estimator Model from Borah, Young, and Curry (1978).

The author selected two bodies of previous research as points of departure in designing the following experiment. Hiltner's (1983) closed-loop otolith assessment procedure showed that subjects seated in a linear sled could perform velocity nulling tasks in the presence of pseudo-random disturbances. Zacharias (1977) tested visual-vestibular interactions in angular motion. Using uncorrelated sum-of-sines inputs, he was able to separate the effects of scene and body rotation. We have learned from their past experiences in order to draw from their successes and, hopefully, to avoid their mistakes.

Our Experiment

The MIT Linear Sled served as the test platform for this experiment. The Sled was modified slightly to incorporate on it the HMD and G-Seat (see Equipment section). We presented subjects with simultaneous pseudo-random (sum-of-sines) inputs to visual, tactile, and vestibular pathways. Using a joystick controller, the subject attemped to null his perceived motion along his Z- (rostro-caudal) axis. The profiles were selected so that

they would be uncorrelated with eachother, creating a sensory conflict, and forcing the subject to choose the "real" motion from among his senses, or develop some combination thereof. The task is considered closed-loop because the joystick signals are added to the sled motion profile to produce the actual velocity.

Closed v. Open-Loop Tasks

We elected to perform closed-loop tasks for the same reason given by Hiltner; that is, the response magnitudes in an open-loop experiment have only relative significance. Although a subject can be "taught" to indicate magnitude beforehand by establishing response exremes, we felt that this method would allow too much response variability. In contrast, the closed-loop task gives response magnitude immediate and absolute meaning. The drawback is that subjects will at times fail to complete a trial because they reached a track limit. Like Hiltner, we ran numerous tests to develop profiles that had high degrees of completion.

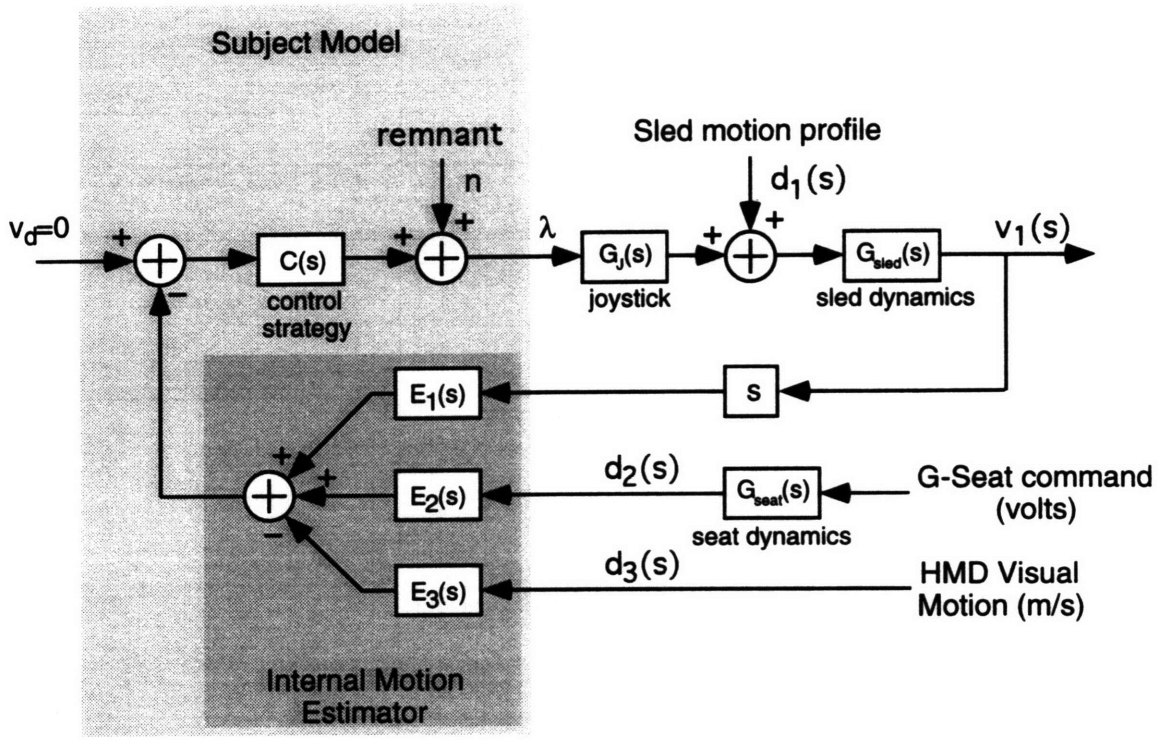


Figure 4-1 Model of system including operator

Disturbance Selection

We used sum-of-sines disturbances as inputs for several reasons. First, they can be made seemingly random from a subject's viewpoint, avoiding problems with predictability. Anticipation by the subject can obliterate any physiological insights that might be gleaned from response data. Secondly, the well-defined frequency content in a sum-of-sines simplifies later analysis. Finally, these profiles are simple to implement in a variety of discrete systems.

We tested subjects over a frequency spectrum of 0.061 to 0.5005 Hz. Although all of the experimental hardware performed well at higher frequencies, during protocol selection we found that subjects were unable to perform the nulling tasks much above 0.5 Hz. At the other extreme, track length limited the use of low frequencies, because the

acceleration detection threshold of approximately 0.005 G mandates high amplitudes (Young (1984)). Moreover, there are sensory performance considerations that impose certain restrictions, as detailed in the Background section.

Hiltner used a similar frequency band in his research. To select specific sinusoids, we followed his method of taking prime multiples of a low base frequency. These are listed in Table 4-1. We faced the added complication of creating separate waveforms for simultaneous application to sled, visual scene, and G-Seat motion. Therefore, we split the 12 frequencies listed into 3 interleaved groups, labelled here as A, B, and C. By presenting the subject with various combinations of these groups, we were able to test the entire spectrum. A sample waveform is presented in Figure 4-2.

		Base Freq:	0.0122 Hz	T= 81.92 s	
A	Prime multiple:	5	11	19	32
	Freq. (Hz)	0.0610	0.1343	0.2319	0.3906
B	Base * mult:	7	13	23	37
	Freq. (Hz)	0.0854	0.1587	0.2808	0.4517
C	Base * mult:	9	17	29	41
	Freq. (Hz)	0.1099	0.2075	0.3540	0.5005

Table 4-1 Disturbance frequencies tested and waveform groupings.

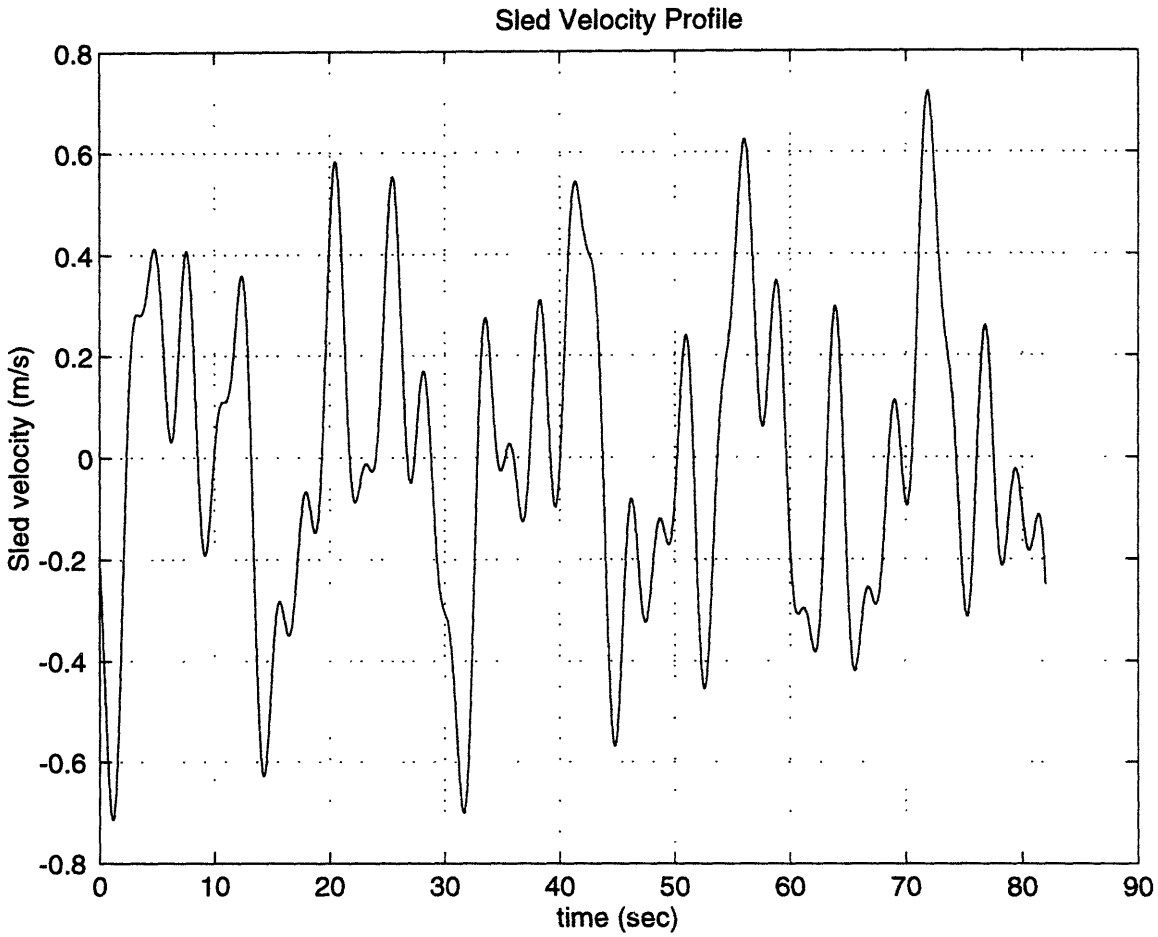


Figure 4-2 Typical disturbance waveform.

In choosing amplitudes for the individual sines, Hiltner went through a long iterative process. His goal was to minimize the amount of time that sled acceleration levels fell below the perception threshold mentioned above (0.005 G). We adopted the values which he arrived at for his final practice and data runs. In deciding the sine phases, we performed our own iterative procedure in MATLAB. The script sought out a phase offset to apply to each succeeding sine component that would minimize the mean sled velocity during the first 2 seconds of a trial. The rationale behind this decision was to increase the likelihood that subjects could complete runs without hitting a limit stop. Furthermore, large initial velocities tended to discomfort the subjects. The resulting waveforms are

giving in Table 4-2. The phase offsets were included in the profiles of both the sled and the HMD, but not in those going to the G-Seat. Each trial ran for 81.92 s, a balance between sufficient data points for analysis versus probability of run completion and subject comfort.

During evaluation of the profiles, we determined that the visual velocities were too low using the given amplitudes. Some of the smaller motions seemed jerky and artificial through the HMD. We attributed this to the limited vertical resolution of the goggles. To compensate, we doubled the waveform amplitudes in the visual motion profiles. The resulting motions appeared more compelling. This change should have no effect on the analysis of the results, as long as it is reflected in the gain calculation for the visual response.

Group	Amp	Freq	$\phi°$	Amp	Freq	$\phi°$	Amp	Freq	$\phi°$	Amp	Freq	$\phi°$
A	0.13	0.0610	0	0.11	0.1343	295	0.10	0.2319	230	0.07	0.3906	165
B	0.13	0.0854	0	0.12	0.1587	65	0.11	0.2808	130	0.07	0.4517	195
C	0.12	0.1099	0	0.11	0.2075	0	0.09	0.3540	0	0.06	0.5005	0

Table 4-2 Disturbance Waveforms, Grouped

Number of Trials and Order of Presentation

With the disturbances now defined, we focused on creating an experimental protocol, or plan, that would allow the greatest number of disturbance combinations. We also wanted to account for possible order or learning effects. However, limited subject endurance (encompassing fatigue, discomfort, and boredom) prevented us from employing a complete Latin Square design. Limiting a session to 12 data trials (after several practice runs, see below), we came up with the matrix given in Table 4-3.

Trial #	Sled	G-Seat	Display
1	A	B	C
2	B	A	C
3	C	B	A
4	B	C	A
5	A	C	B
6	C	A	B
7	C	B	A
8	B	A	C
9	A	C	B
10	C	A	B
11	A	B	C
12	B	C	A

Table 4-3 Number of Trials and Order of Stimulus. Presentation by Modality

Practice Runs

Before collecting data, we allowed each subject to perform several practice runs, to familiarize him or herself with the equipment and task at hand. This involved a run or two of using only the Sled and joystick in an eyes open condition, followed by practice trials using all three stimuli. These runs employed frequency groupings not used in the data runs (i.e. combinations other than the A,B, and C groups of Table 4-3). Practice continued until the subject completed a run and felt comfortable with the task.

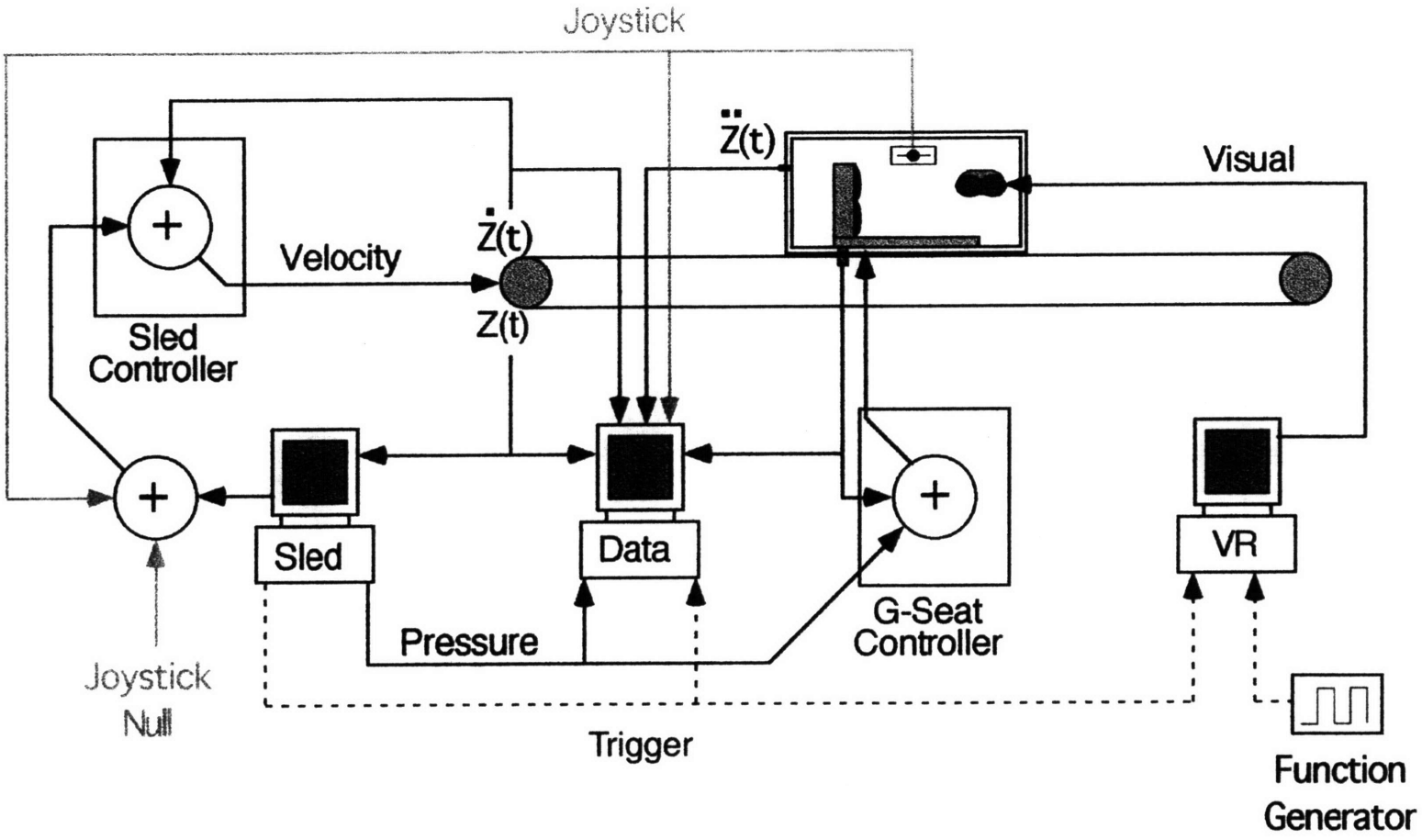


Figure 4-3 Schematic of hardware and of signal flow.

Subject control : joystick gain and filter

A new joystick was designed and built for this experiment. It was a long (approximately 8") travel, slider-type controller. We followed Hiltner's lead in low-pass filtering the joystick signal before summing it with the Sled disturbance, in an effort to avoid operator control problems. The Sled program provides for digital filtering and gain control of the joystick signal. We set the filter pole to have a time constant of 0.095 s. While examining the joystick output, we discovered an undesirable amount of 60 Hz line noise in the signal. To correct this, we placed a Krohn-Hite Model 3340 analog filter, set at 0 dB gain, max-flat low-pass, and a break frequency of 40 Hz upstream of the computer. After experimentation, we also settled on a joystick gain that let maximum displacement of the joystick equal 90% of the maximum motion disturbance velocity. We found that higher joystick gains led to subject induced oscillations and resultant instability.

Due to internal friction the joystick did not always return to its exact center position before starting a trial. We became concerned that an unexpected initial joystick commmand could cause the sled to lunge. Therefore, as a precaution, we ran the joystick command through a summing circuit before it reached the Sled computer. Before each run, the experimenter could zero out any unwanted bias. Because the control task is closed loop, and the adjustments were small compared with commanded voltages, we did not anticipate any problems with this approach. Furthermore, our analysis techniques were not sensitive to small DC trends in signals.

Experimental Setup and Signal Flow

Figure3-3 is a schematic representation of our experimental setup, and of the flow of commands and data among the various pieces of equipment. Detailed descriptions of each device are given in the Equipment section.

Experiment Checklist

The following is a checklist of tasks performed before, during, and after each session.

I. Before Subject Arrival

A. Power-up computers, power supplies, and amplifiers. Open valve on air supply.

B. Maintenance and safety checks

1. check limit switches on Sled track
2. adjust Sled position pot as needed
3. check joystick function

II. Subject Arrival

A. Describe experiment and obtain informed consent.

B. Perform vestibular function assessment, if unknown.

C. Help subject into sled, fasten harness, and adjust head restraint. Familiarization with controls and safety precautions.

D. Audio check.

III. Subject Practice

A. Sled only, with eyes open. Practice control task.

B. (When ready) Subject dons HMD. G-Seat powered up. Harness and head restraint adjusted.

C. Practice runs with full equipment. Continue until a run is completed and subject is ready to continue.

IV. Each Experimental Run

A. Enable sled and move to proper starting position on track.

B. Zero joystick as necessary.

C. Set visual display to starting point, awaiting trigger

D. Open data collection file on data computer, await trigger

E. Comfirm that subject is ready

F. Start Sled and G-Seat profiles. Other computers are triggered automatically.

E. Turn off experimenter VOX audio circuit to minimize noise.

F. (If subject hits or nears limit switches) Stop run and disable Sled. Cancel display motion profile and data collection. Inform subject that he reached a track end, but not which one. Recenter Sled and reset profiles. Repeat run.

G. (If run is completed) Profiles terminate automatically. Inform subject that the run was completed and close data file. Prepare for next run.

V. Post experiment

A. Disable Sled

B. Assist with subject egress.

C. Subject debriefing

D. (If necessary) Prepare for next subject. Otherwise, shut down equipment.

Results and Analysis

Six subjects volunteered for the experiment detailed in the preceeding section. Four female and two male students took part. All were healthy and exhibited normal vestibular function. Every subject, except for the first, completed all 12 planned trials. Subject #1 (DH) was not able to repeat runs in which a track end was reached, because we had not yet configured the computers to allow repititions of a trial. After equipment modifications, all the other subjects were able to repeat trials as necessary. In all cases, a subject repeated a trial at most twice before completing it. This represents a large improvement over Hiltner's difficulty in obtaining complete runs. We expect that our use of fewer sinusoids (4 versus 12) in a given waveform was the major reason that completion rate rose. Nevertheless, we feel that these profiles were sufficiently above thresholds, and random enough to be unpredictable.

Data from all the experiments were transfered via FTP to a Apple Power Macintosh 7100/80. The analysis and graphs in this section were done using MATLAB v4.2 .

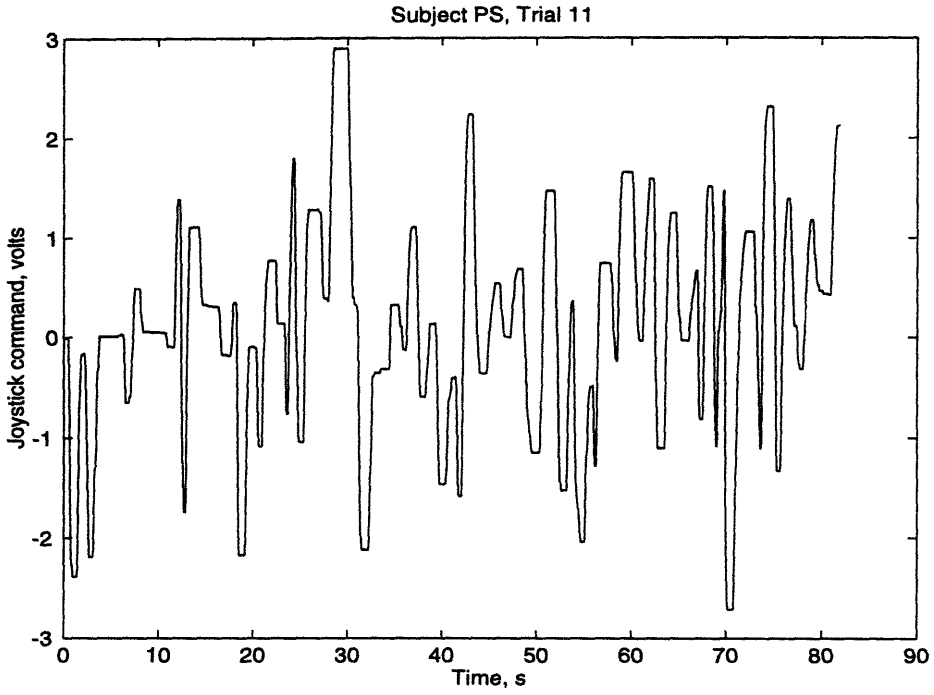


Figure 5-1 Time history of joystick signal. Plateaus away from joystick center suggest the subject hesitated during periods of uncertainty.

Figure 5-1 is a typical plot of the subject's controller output as he attempts to null his perceived velocity. Several key characteristics of this graph were common among subjects. The rapid reversals, with few smooth transitions, are associated with "bang-bang" controllers. (Young and Meiry (1965)) Although the joystick was designed for continuous output and smooth transitions, it was predominantly not used this way.

The flat segments, which indicate that the subject held the controller still, are problematic as well (and suggest that uncertain perception underlay the bang-bang approach). We interpret these segments as evidence that the subjects adopted a "wait and see" strategy during periods in which they were uncertain about their motion. Once the subject had regained a mental picture of his motion, he would overcorrect in order to "catch up."

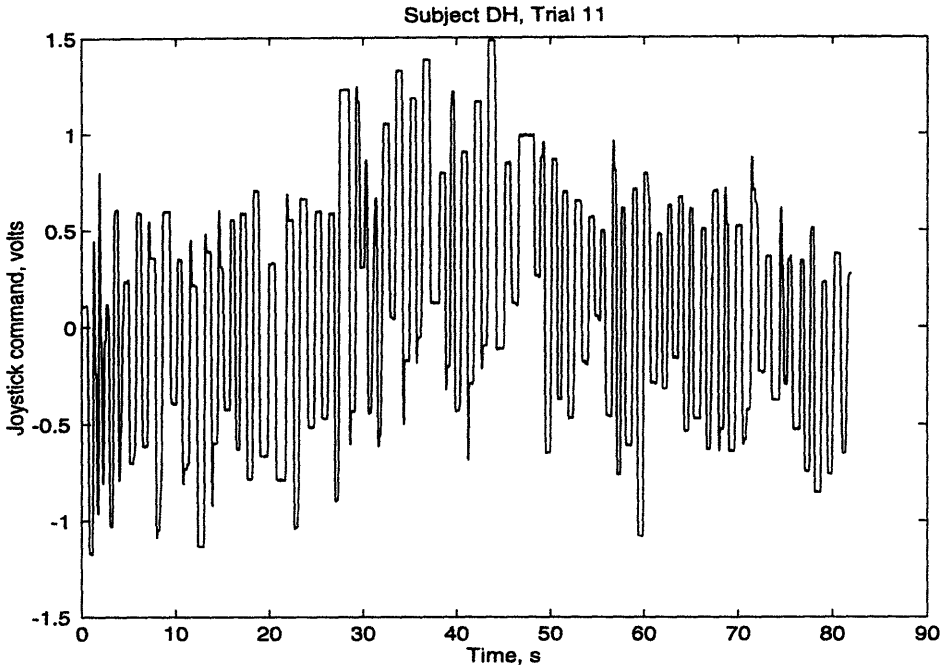


Figure 5-2 Time history of joystick signal. Rapid reversals (0.7 - 0.8 Hz) above the highest disturbance frequency (0.5005 Hz) may indicate injected dither on the subject's part, or possibly a response to a disturbance harmonic.

A final point of interest in the joystick output can be seen in Figure 4-2. In this case, the subject executed large control amplitudes at frequencies higher than any of the disturbance inputs. Two possible explanations are proposed for this behavior. The subject may have been "dithering" her control signal at a high frequency. Dithering is the intentional injection of noise into a signal in order to raise the signal out of a sub-threshold dead band. It is possible that the subject was adding dither to obtain supra-threshold vestibular stimuli on which to base her perception of motion. Another possibility is that the subject was responding to a harmonic of a disturbance signal.

During post-session interviews with the subjects, it became clear that they considered the sled disturbances to be the only "real" motion cues. As a result, they consciously discounted the visual and tactile stimuli. The subjects had been explicitly instructed beforehand to rely on all their senses, so this revelation was discouraging. Nevertheless,

we proceeded with analysis of the data, hoping that cognitive processes played only a part in their sensory integration, and that the visual and tactile cues were processed subconsciously.

Our analysis of the data followed two distinct paths. The first consisted of a frequency-based examination of the subjects' responses. We used these results to construct a linear model of human perception of motion, resulting from multiple sensory inputs. The second was a time-based breakdown of the data. We felt the two approaches would each contribute to the total picture. They are described below.

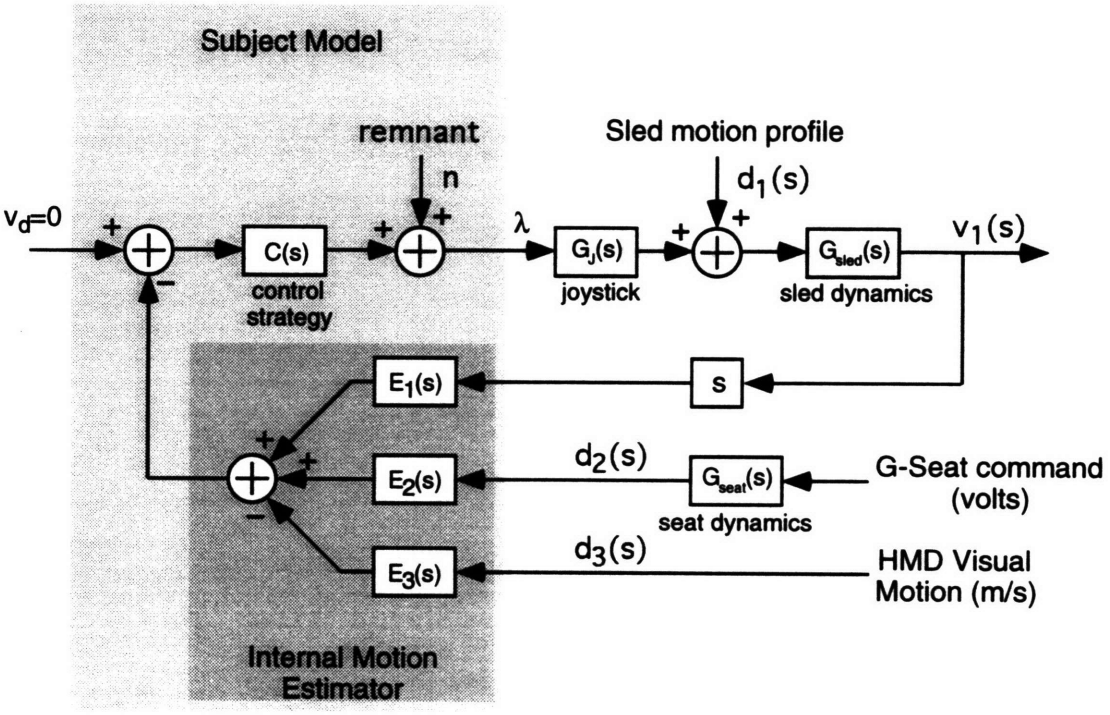


Figure 5-3 Block diagram model of experiment.

Frequency Analysis

Approach

We drew upon the work of Zacharias (1979) in constructing the method for deriving transfer functions from the data. Figure 5-3 shows the block diagram model of our experiment and of the subject. A disturbance input (sum-of-sines) is passed to each device. The three sensory channels we used, vestibular, tactile, and visual, are each represented by a single estimator. The inputs to the estimators are the appropriate stimuli for that modality: acceleration (vestibular), pressure (tactile), or velocity (vision). The estimator outputs are then combined linearly into the subject's overall sense of velocity, which he attempts to null.

Although the subject clearly uses some control strategy in moving the joystick, it is not possible within the confines of the current experiment to isolate this strategy block from the estimators. As such, it remains grouped with the estimators. Because the system is assumed to be linear, the subject's output should consist only of the input frequencies. The operator remnant, then, is defined to be the components of the output that cannot be correlated with the inputs. Hiltner (1983) based the "goodness" of a trial inversely on the magnitude of the remnant.

Our analysis relied heavily on finding the auto- and cross-power spectral densities of the measurable inputs and outputs. This approach worked because the inputs were chosen to be mutually uncorrelated (see Methods Section). We began by finding the system outputs in terms of the inputs:

$$\lambda = \frac{n - C(s)[E_1(s)G_{sled}(s)d_1(s)s + E_2(s)d_2(s) - E_3(s)d_3(s)]}{\Delta}$$

$$v_1 = \frac{G_{sled}(s)(d_1 + G_j(n - C(s)[E_2 d_2 - E_3 d_3]))}{\Delta}$$

where $\Delta = [1 + sC(s)E_1(s)G_{sled}(s)G_j(s)]$

Next we expressed the same equations in terms of auto- and cross-correlations.

$$\Phi_{d_1\lambda} = \frac{\Phi_{d_1 n} - C(s)[E_1(s)G_{sled}(s)\Phi_{d_1 d_1}(s)s + E_2(s)\Phi_{d_1 d_2}(s) - E_3(s)\Phi_{d_1 d_3}(s)]}{\Delta}$$

$$\Phi_{d_1 v_1} = \frac{G_{sled}(s)(\Phi_{d_1 d_1} + G_j(\Phi_{d_1 n} - C(s)[E_2\Phi_{d_1 d_2} - E_3\Phi_{d_1 d_3}]))}{\Delta}$$

$$\Phi_{d_2\lambda} = \frac{\Phi_{d_2 n} - C(s)[E_1(s)G_{sled}(s)\Phi_{d_2 d_1}(s)s + E_2(s)\Phi_{d_2 d_2}(s) - E_3(s)\Phi_{d_2 d_3}(s)]}{\Delta}$$

$$\Phi_{d_3\lambda} = \frac{\Phi_{d_3 n} - C(s)[E_1(s)G_{sled}(s)\Phi_{d_3 d_1}(s)s + E_2(s)\Phi_{d_3 d_2}(s) - E_3(s)\Phi_{d_3 d_3}(s)]}{\Delta}$$

Since the inputs and operator remnant are non-correlated with eachother, the following holds:

$$\Phi_{d_1 n} = \Phi_{d_2 n} = \Phi_{d_3 n} = 0$$
$$\Phi_{d_1 d_2} = \Phi_{d_1 d_3} = \Phi_{d_2 d_3} = 0$$

Therefore, after simplifying, we take the following ratios:

$$\frac{\Phi_{d_1\lambda}}{\Phi_{d_1 v}} = -C(s)E_1(s)s = a_1$$

$$\frac{\Phi_{d_2\lambda}}{\Phi_{d_2 d_2}} = \frac{-C(s)E_2(s)}{\Delta} = a_2$$

$$\frac{\Phi_{d_3\lambda}}{\Phi_{d_3 d_3}} = \frac{C(s)E_3(s)}{\Delta} = a_3$$

Each ratio, a_1 - a_3, can be calculated from the available outputs. Therefore, one can solve these equations simulataneously:

$$C(s)E_1(s) = \frac{-a_1}{s}$$

$$C(s)E_2(s) = -\Delta a_2$$

$$C(s)E_3(s) = \Delta a_3$$

As previously stated, it is not possible to separate out the control strategy. At the frequencies we tested, the sled dynamics can be modelled as a unity gain. Using MATLAB, we wrote scripts to find the *a* ratios, calculate the cascaded control-estimators ($C(s)E_x(s)$), and present the results in Bode plot form. MATLAB provides extensive tools for finding auto- and cross-power spectral densities. In our analysis, we used the default MATLAB settings for data windowing (no windowing). The MATLAB files are given in Appendix B.

Confidence Intervals

As with most experiments in physiology, we designed the protocol to present several repititions of each set of stimuli. By performing 12 trials, each modality (vestibular, tactile, visual) had four repititions at each tested frequency. Unfortunately, establishing confidence intervals proved to be very difficult, largely because of the manner in which the ratios (which are complex numbers) are used to derive the estimators.

Using ellipses to define the variance of a population of complex numbers is a common technique, and the resulting covariates are not independent of each other (Figure 5-4). This was a problem in deciding how to propage the errors through the quotients given

above. As a result, we felt compelled to use an overly conservative method to determine confidence intervals. The rectangle surrounding the data points in the figure represents our attempt at a solution. The figure shows a typical case: relatively well behaved data that can be enclosed only be a large rectangle that contains the origin. The resulting variances are larger than the mean. Upon conversion back to magnitude and phase, the lower bound becomes negative, causing that half of the error bar to jump off the plot, as evidenced by the lack of lower error bars for some data points.

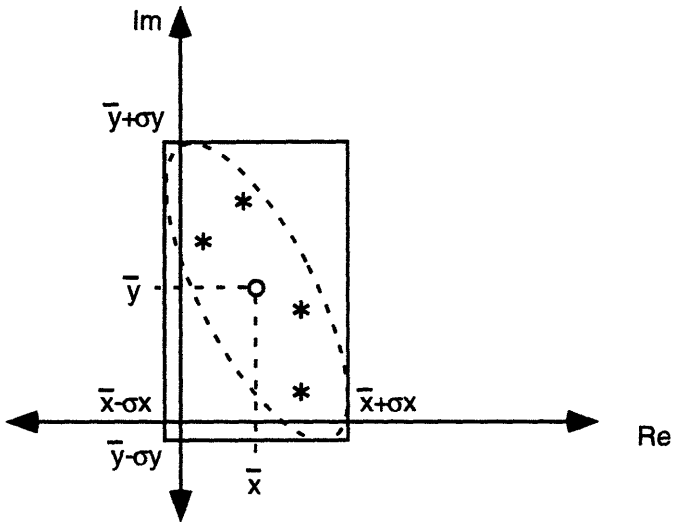


Figure 5-4 Responses to repeated conditions show typical scatter. Asterisks (*) are the *a* values. Conservate error estimates at times resulted in problems with overly large variances.

Results

The following pages contain the results for all six subjects, represented as Bode plots. Three plots were generated per subject; one each for vestibular, tactile, and visual pathways. They represent the response of the cascaded estimator and control strategy blocks of Figure 5-3. The plots may be interpreted as follows:

Vestibular $C(s)E_1(s)$ The portion of the subject's output (as recorded with the joystick) that is correlated with Sled acceleration. We based the analysis on

actual Sled velocity rather than from the Sled disturbance signal, in order to obtain a clearer picture of the stimulus as experienced by the subject. An additional concern was that there would be some phase lag between the disturbance and actual Sled motion. Differentiation (in order to obtain a relationship between input acceleration and output joystick velocity) was performed in the frequency domain, as indicated in the equation derivations above.

Gain and phase are calculated from subject commanded velocity over input acceleration. The gain is proportional to m/s over m/s^2, scaled by the joystick gain. The model predicts that an integration of acceleration back into velocity occurs in the estimator block.

Visual $C(s)E_2(s)$ This is subject's response to visual stimuli. The magnitude is in velocity / velocity, scaled by the joystick. Visual velocity is linear, rather than angular. However, because all motion was constrained to the subject's Z-axis, which was parallel with the virtual ceiling, the ratio between linear and angular velocity remained constant.

Tactile $C(s)E_3(s)$ Response to tactile (G-seat) stimuli. In this case, the input is proportional to *pressure*, as recorded by the transducers in the seat. The assumption is that pressures are interpreted as accelerations, in keeping with our understanding of the G-seat's cueing method. Again, the estimator must perform an integration in order to output a velocity estimate.

Figure 5-5

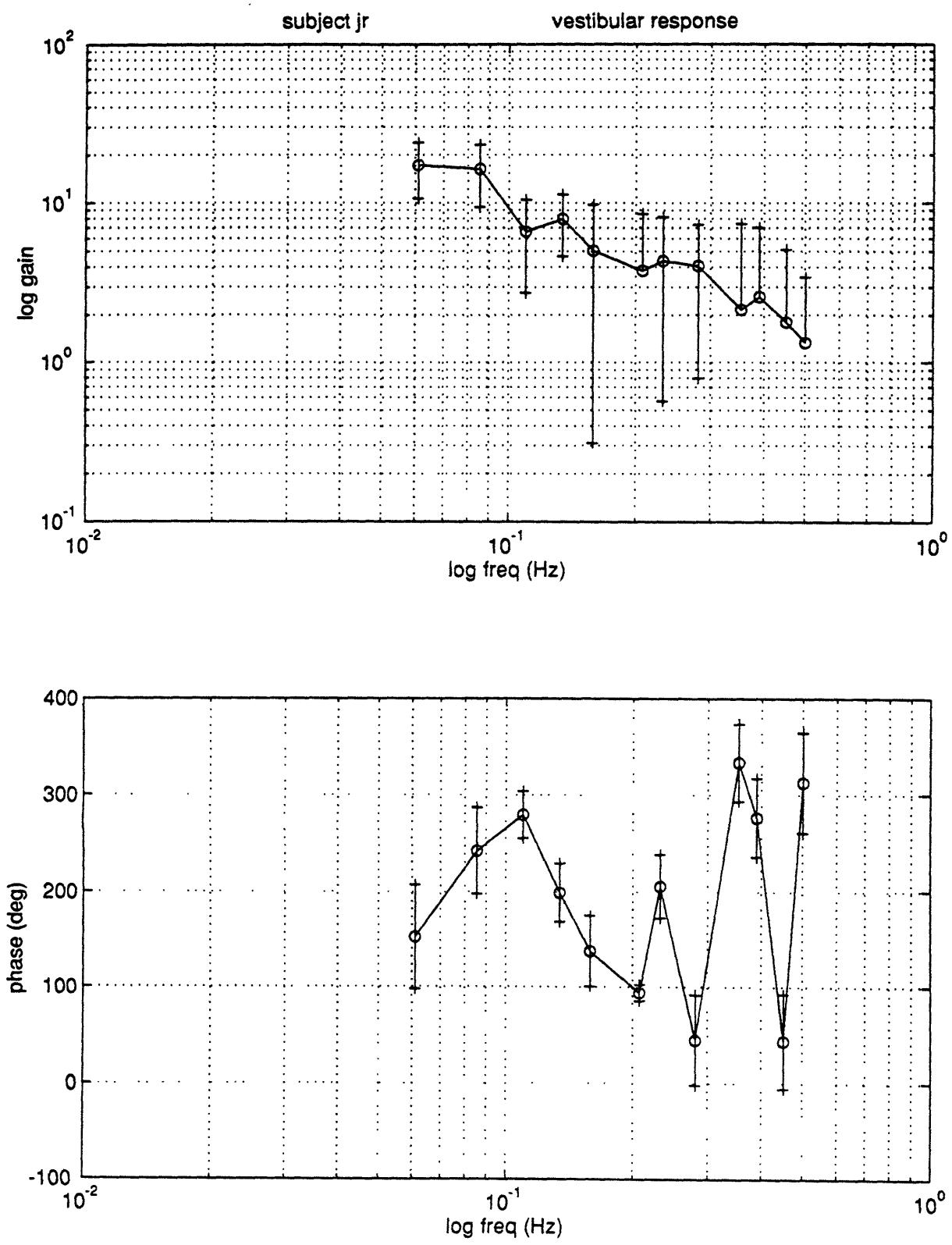
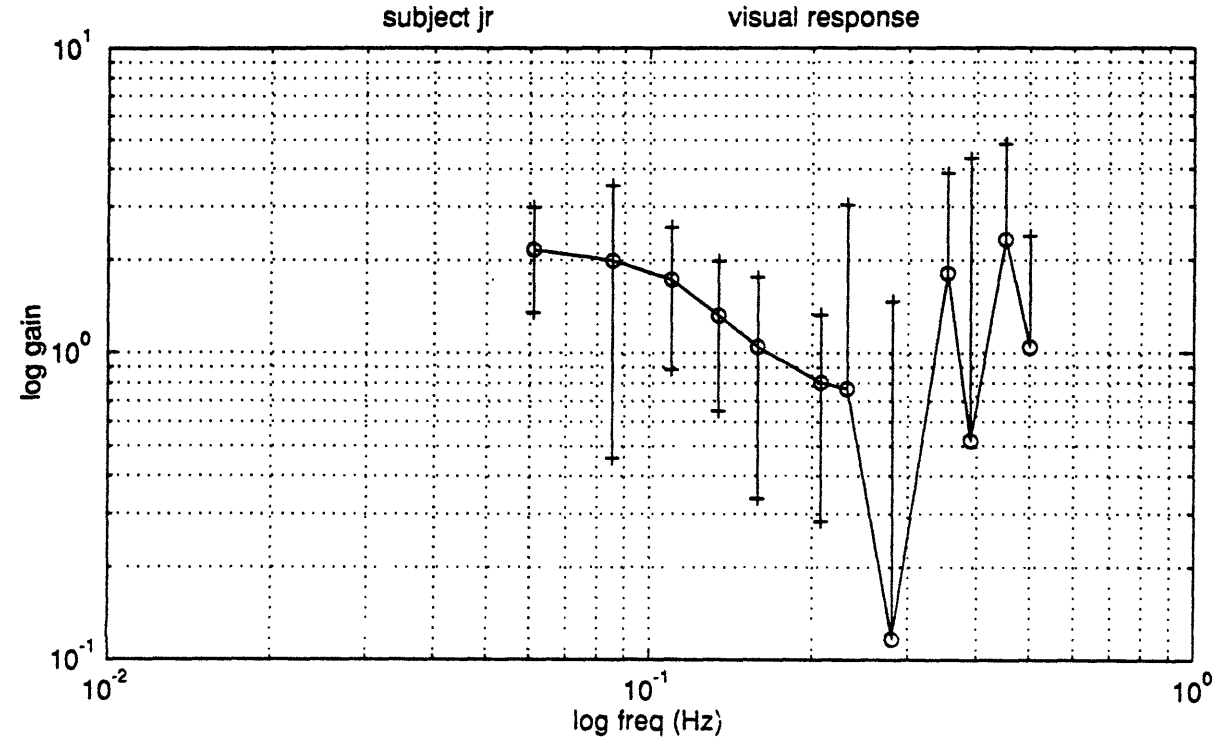

subject jr
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-6


subject jr
visual response
log gain
log freq (Hz)

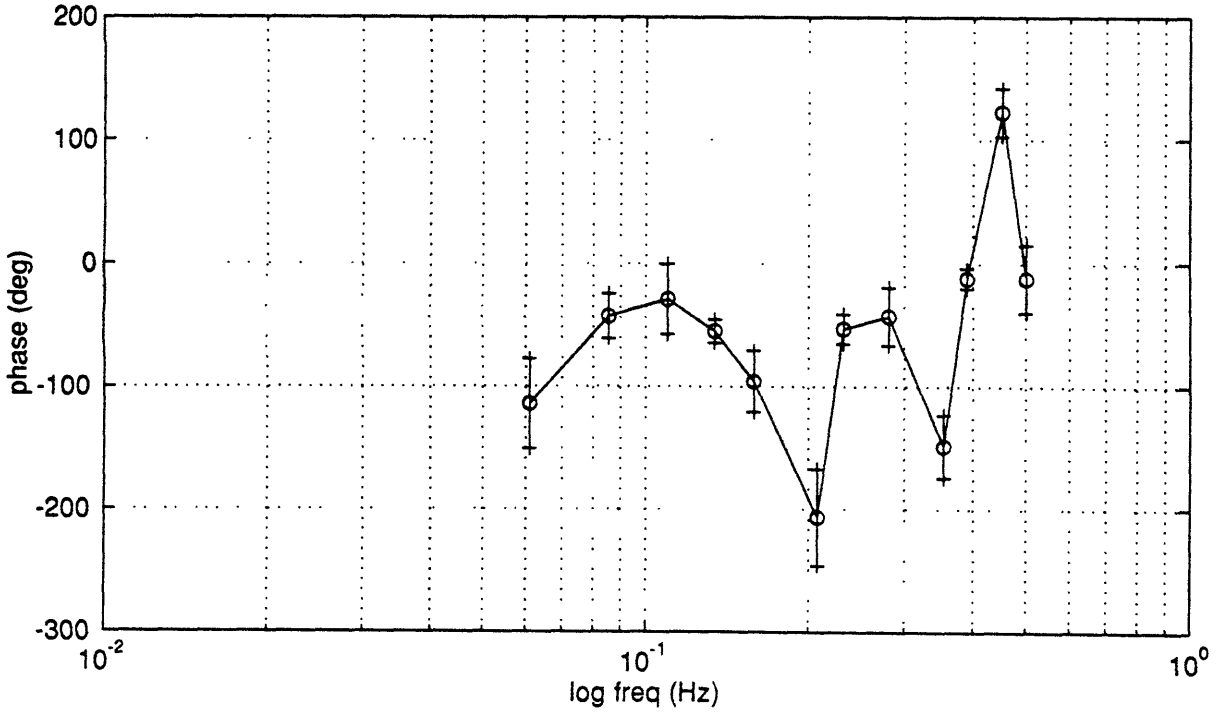

phase (deg)
log freq (Hz)

Figure 5-7

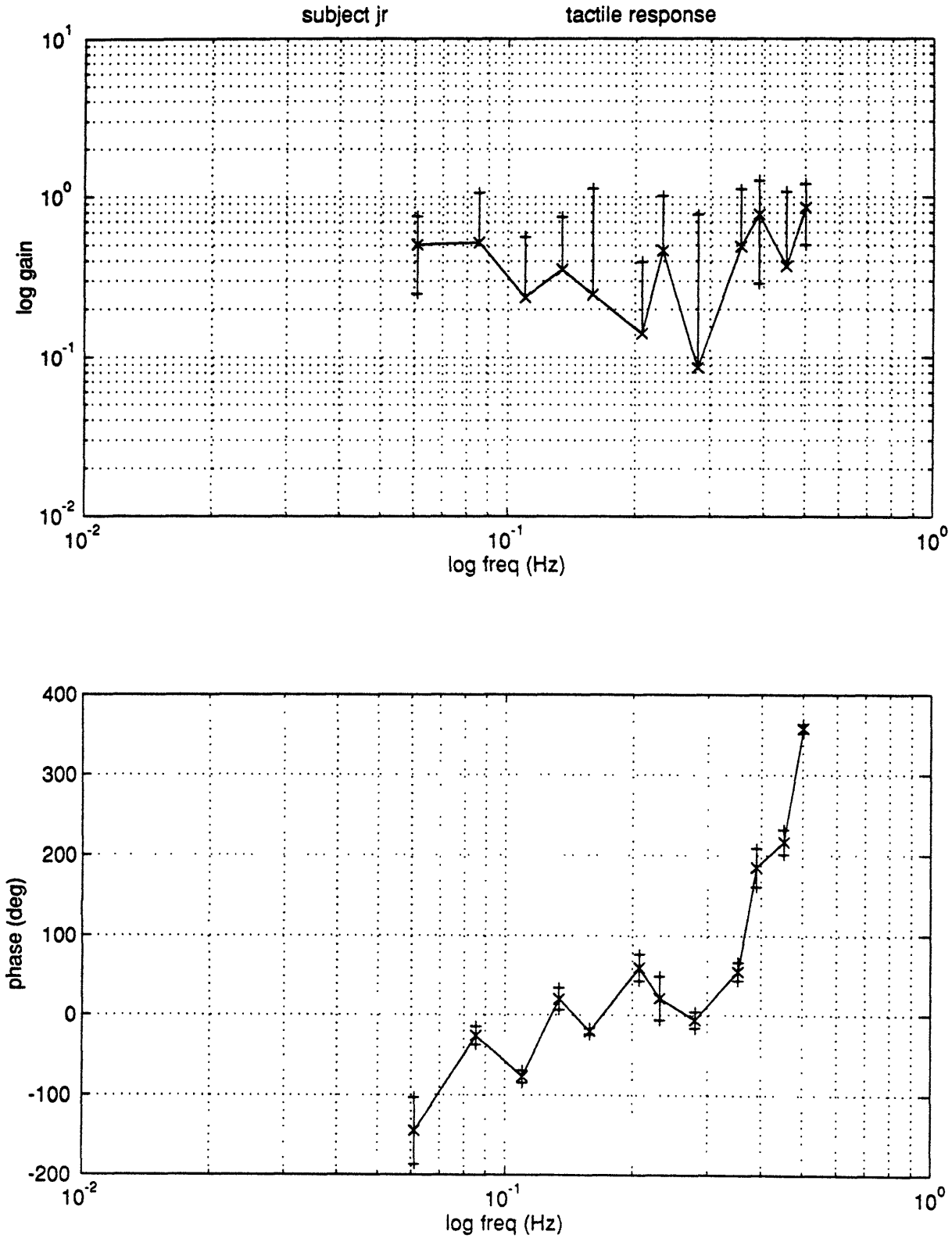

subject jr
tactile response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-8

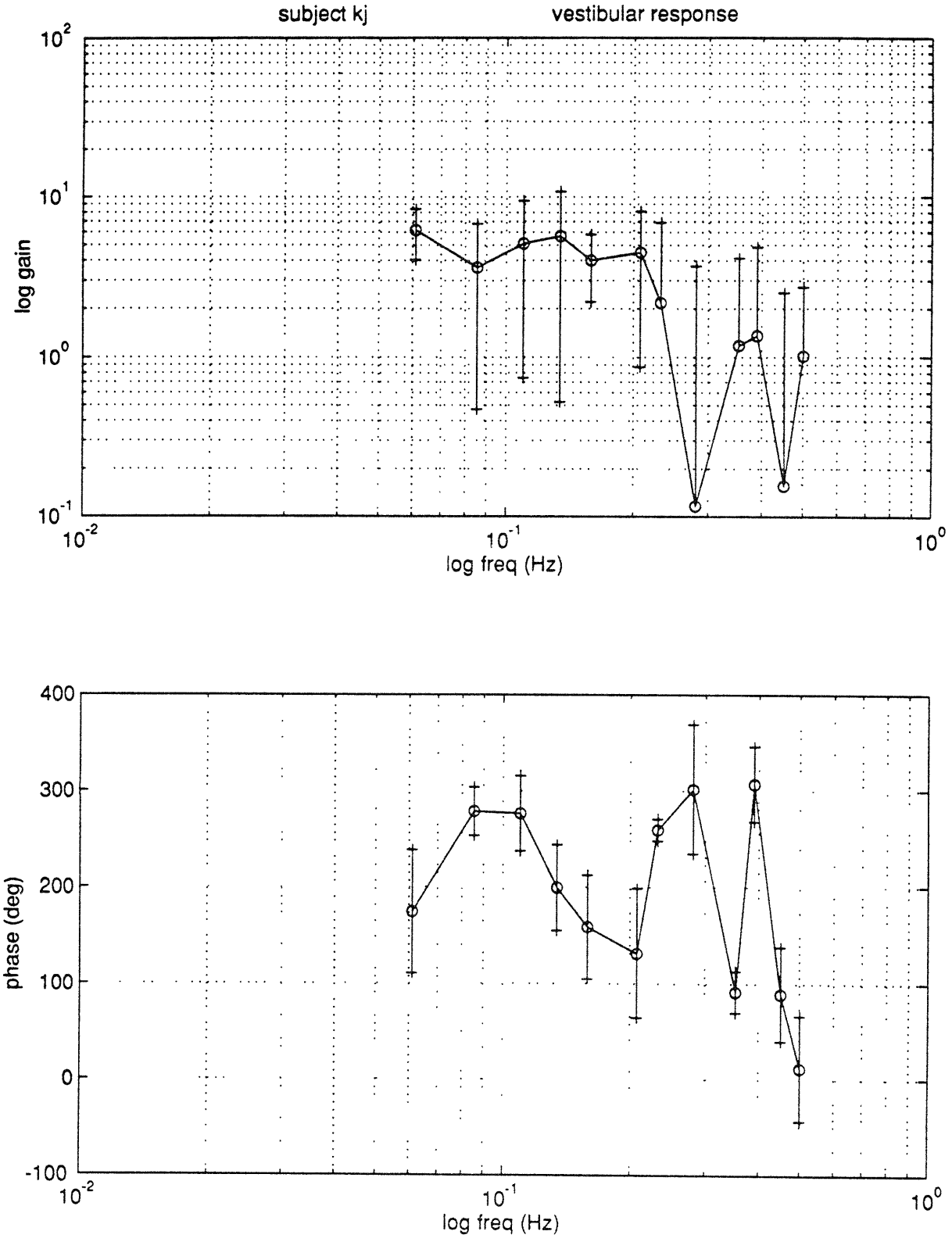

subject kj
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-9

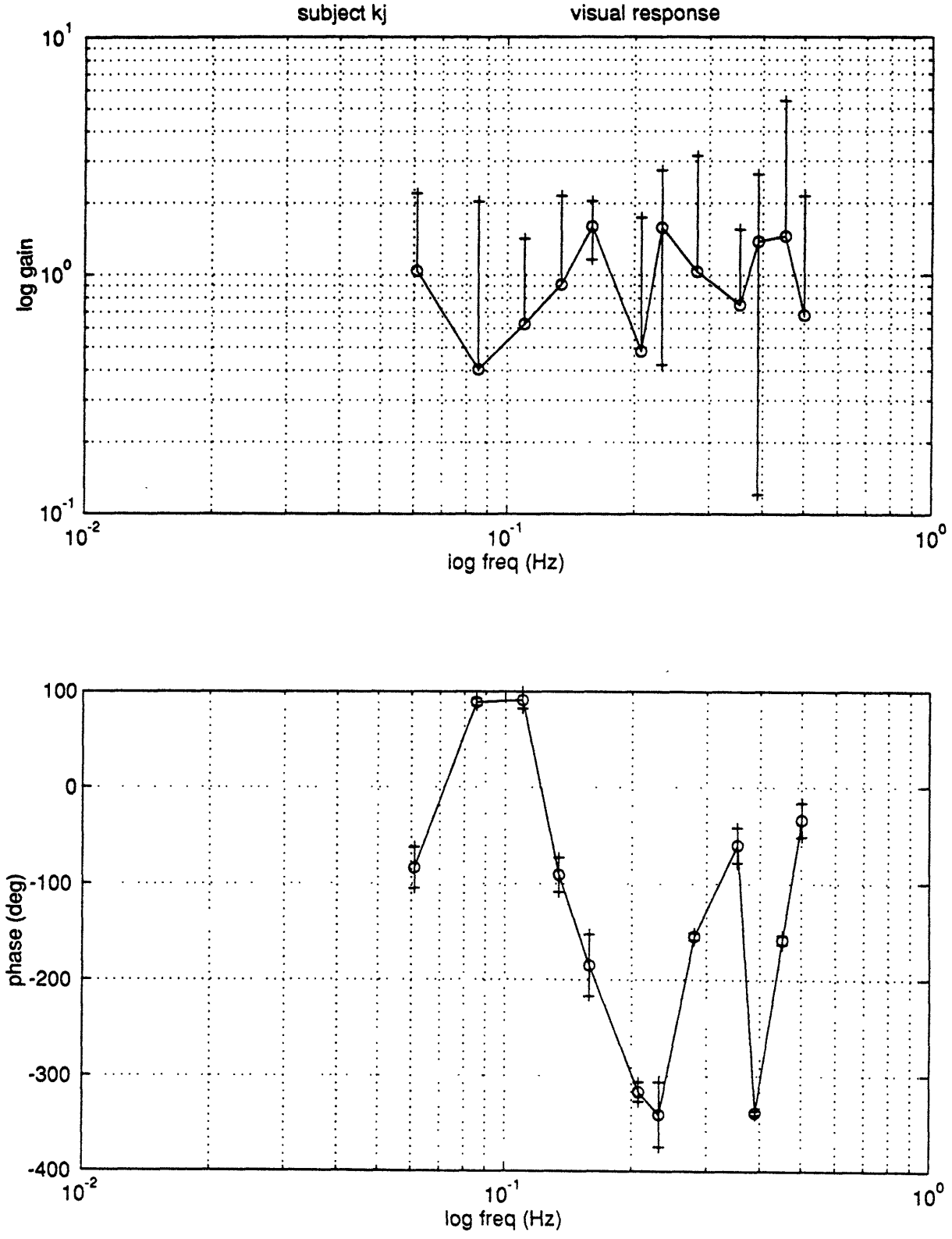

subject kj
visual response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-10

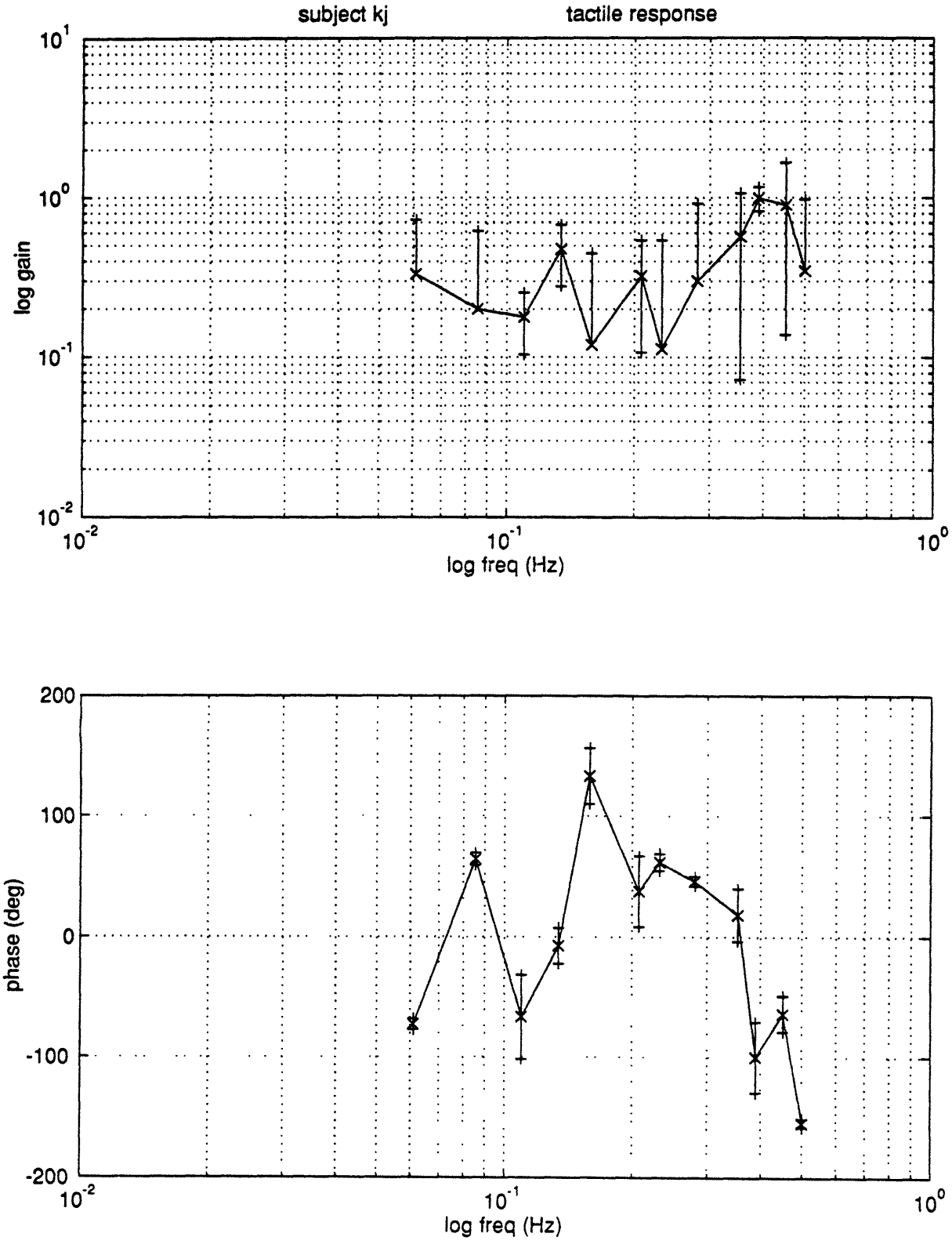

subject kj
tactile response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-11

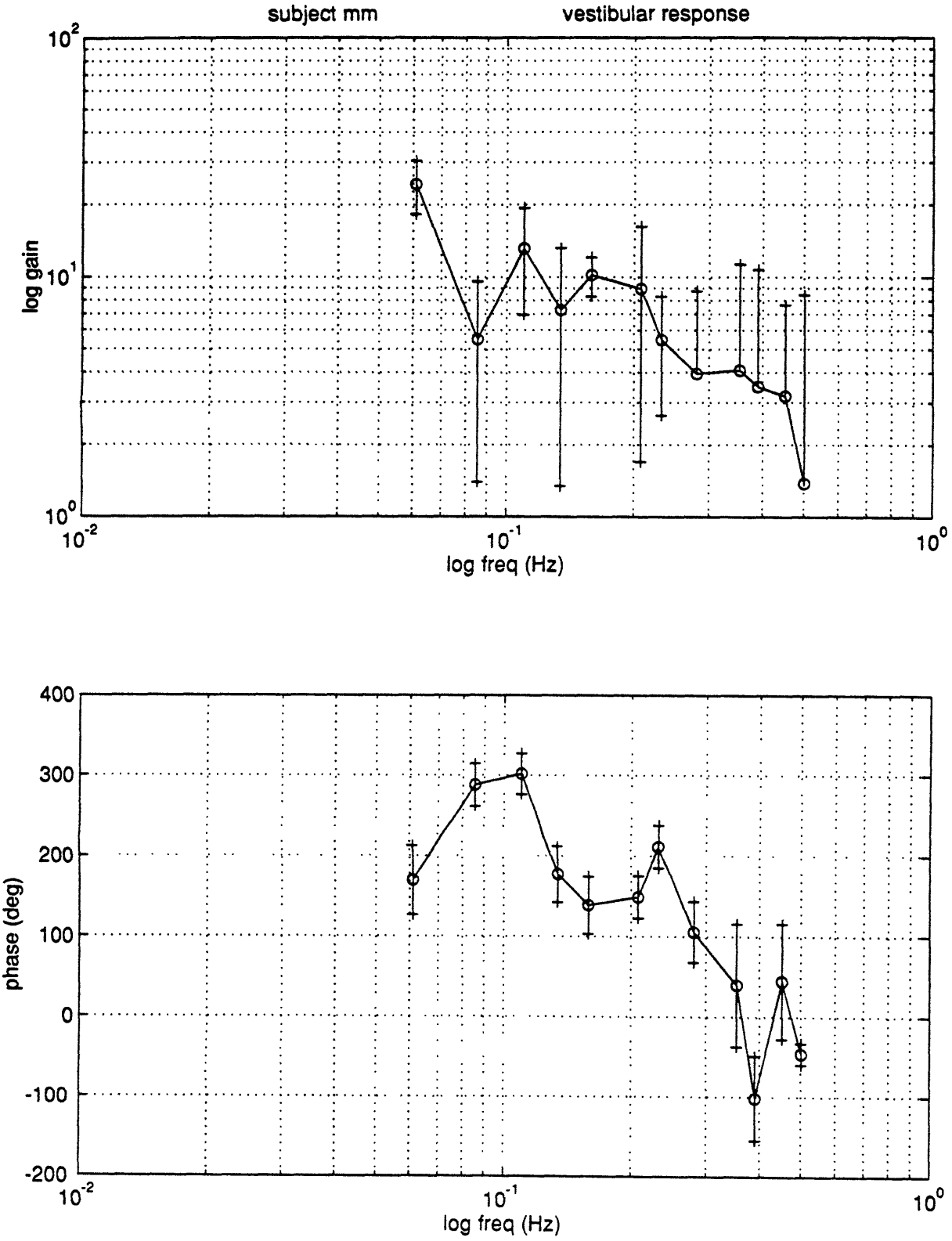

subject mm
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-12

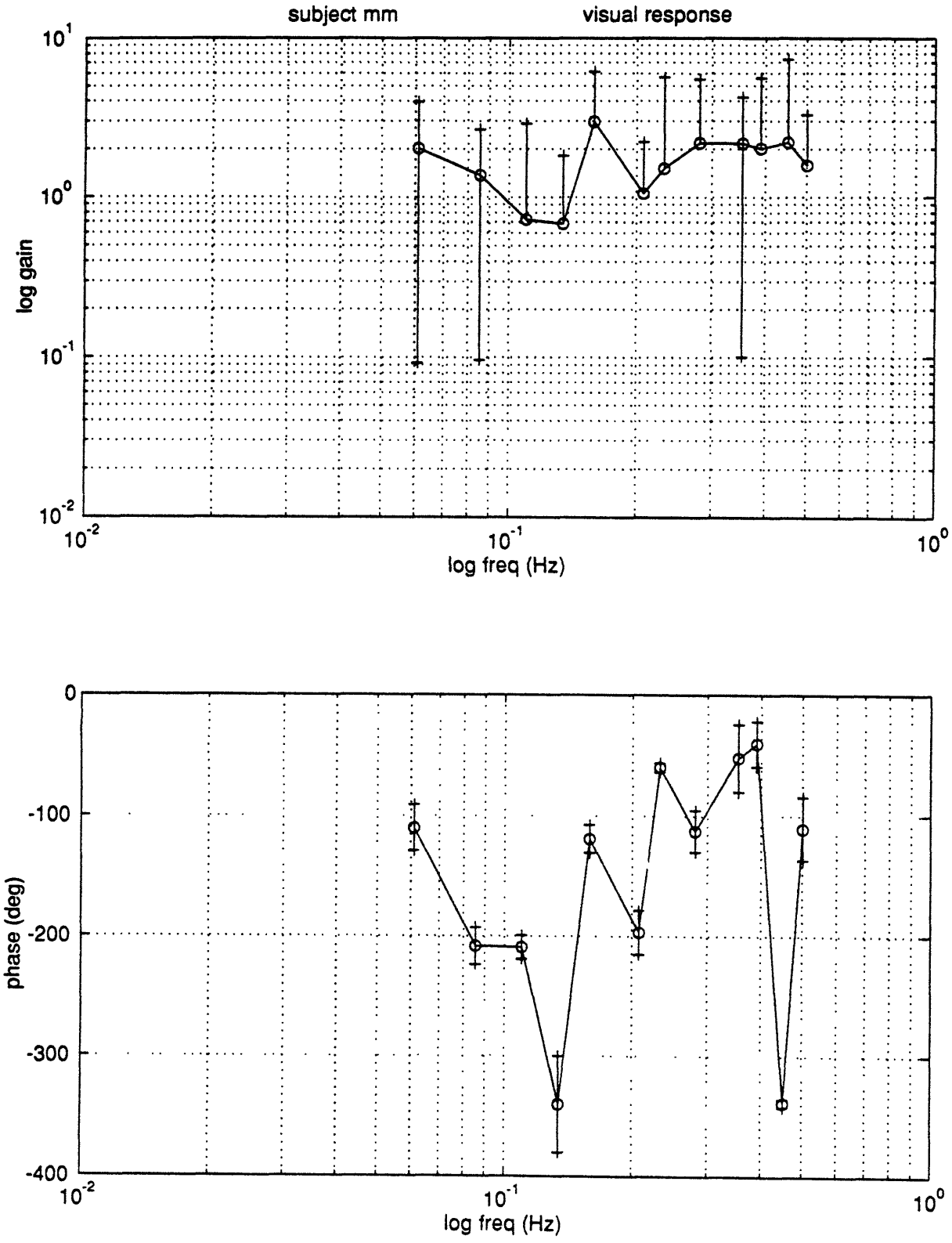

subject mm
visual response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-13

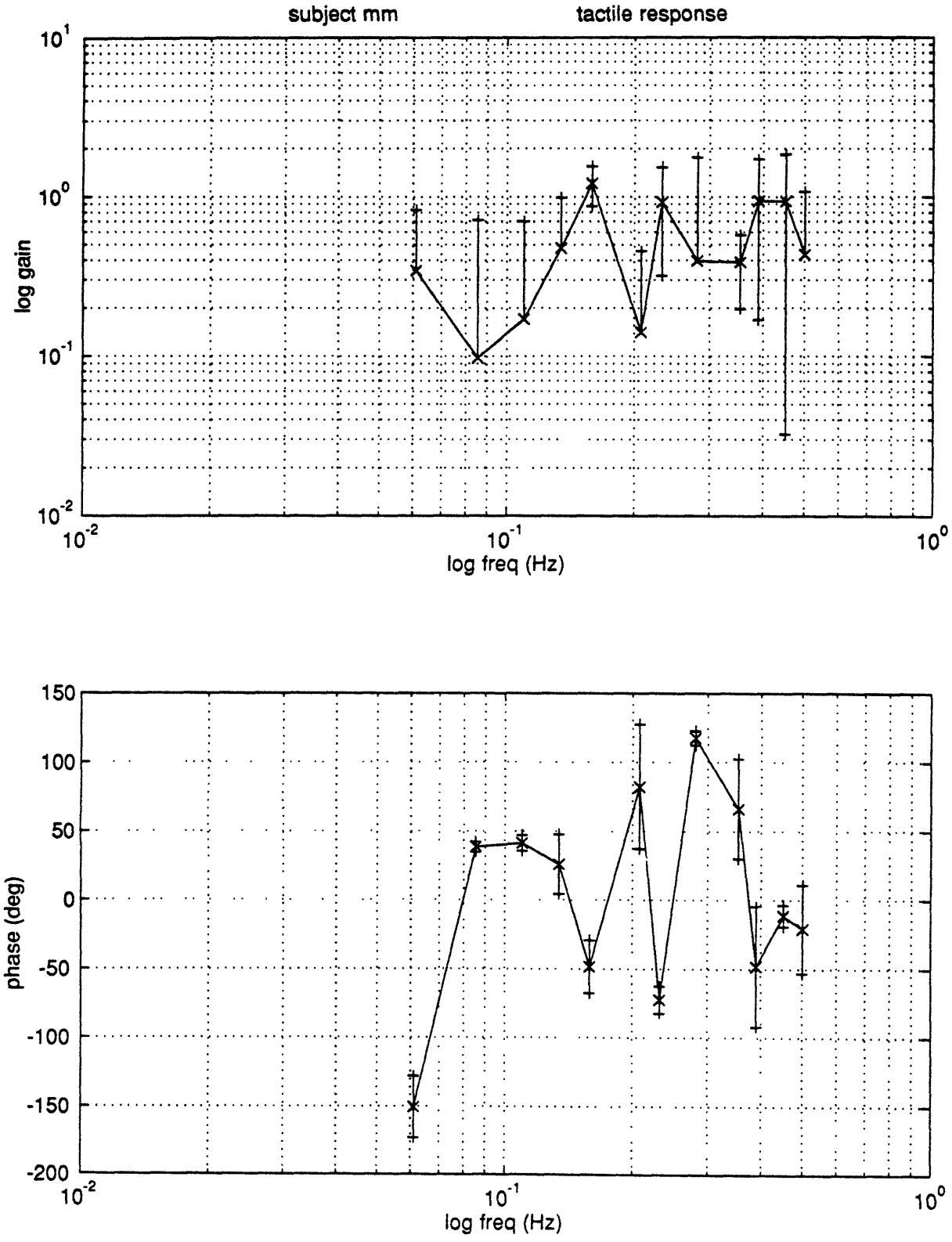

subject mm
tactile response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-14

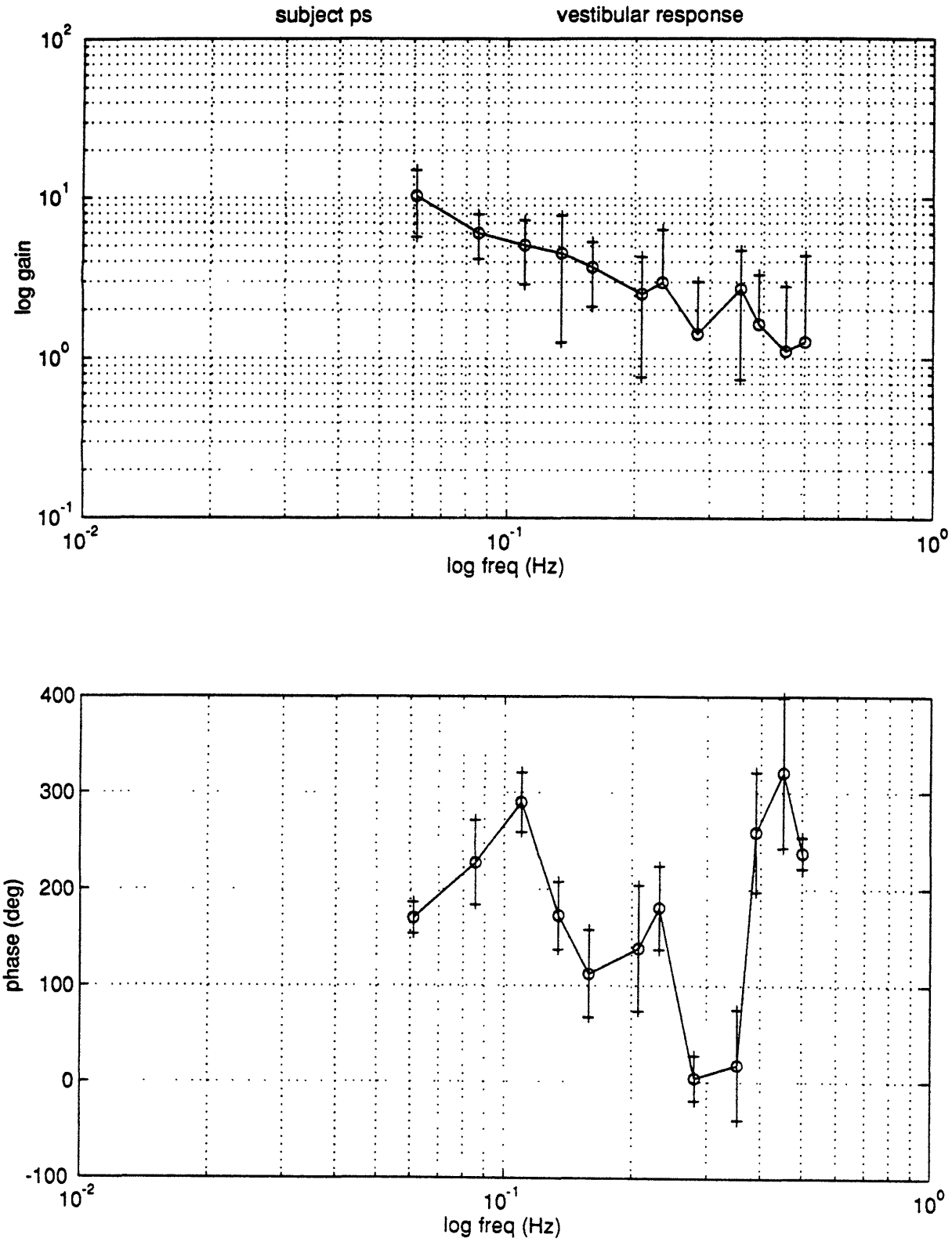

subject ps
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-15

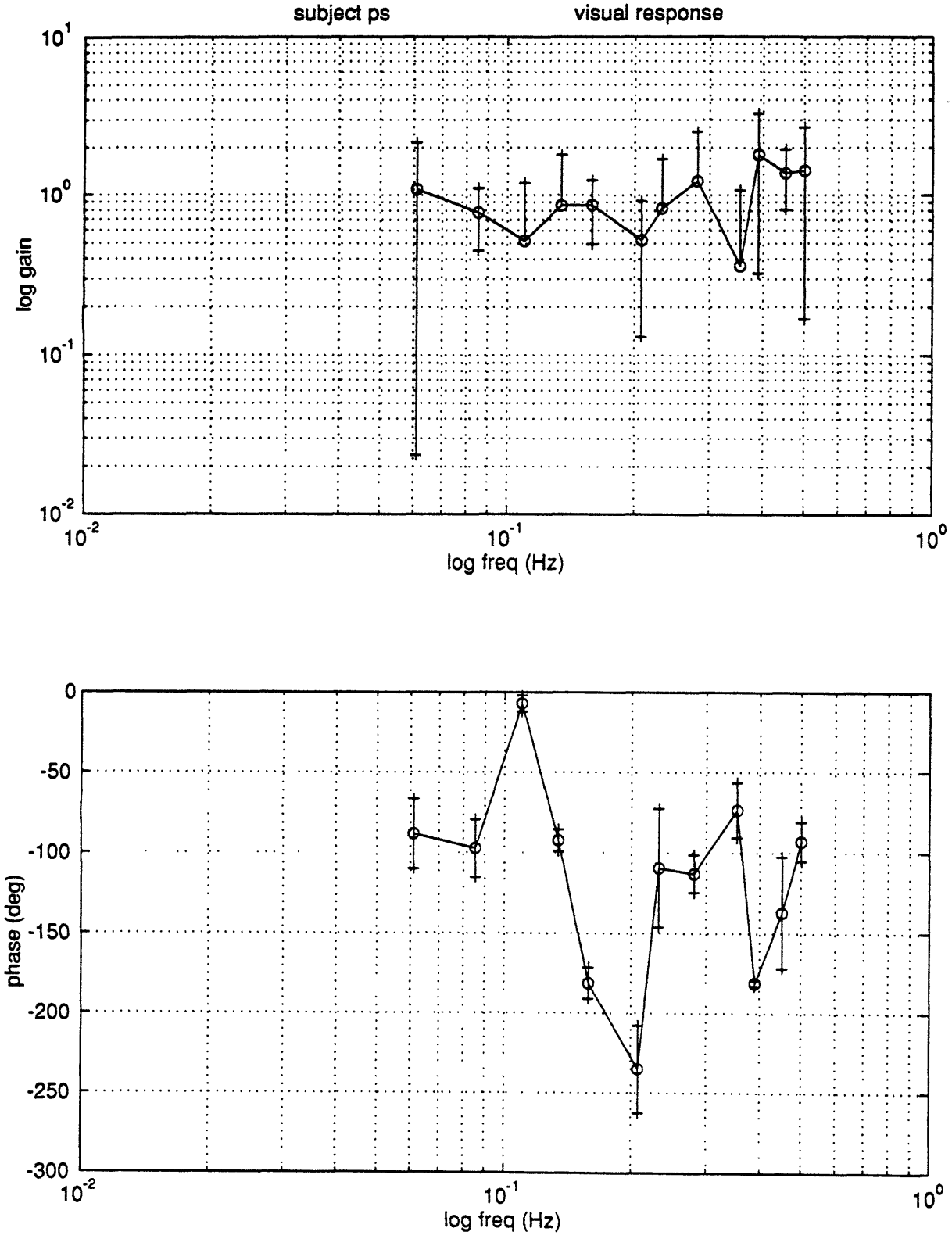

subject ps
visual response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-16

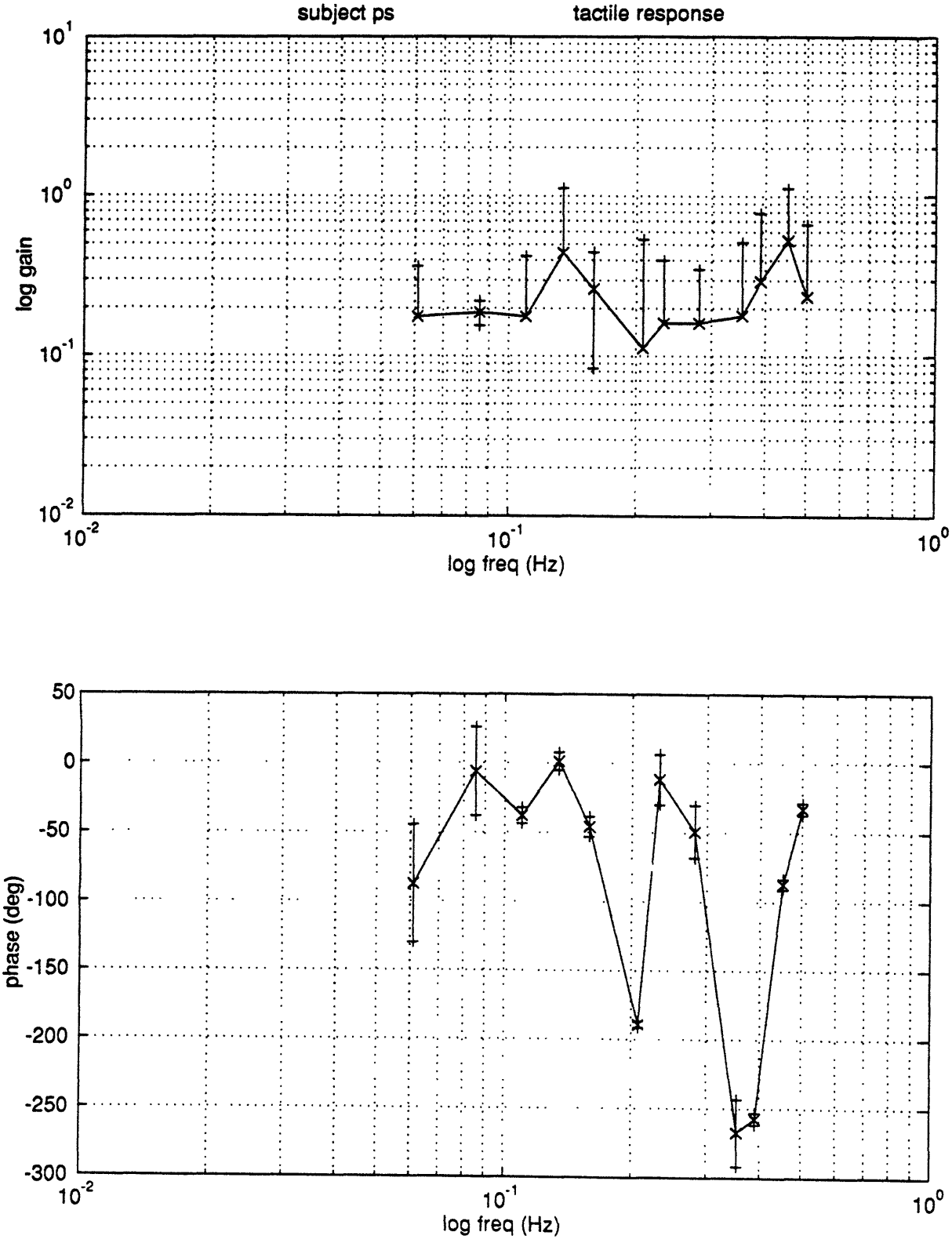

subject ps
tactile response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-17

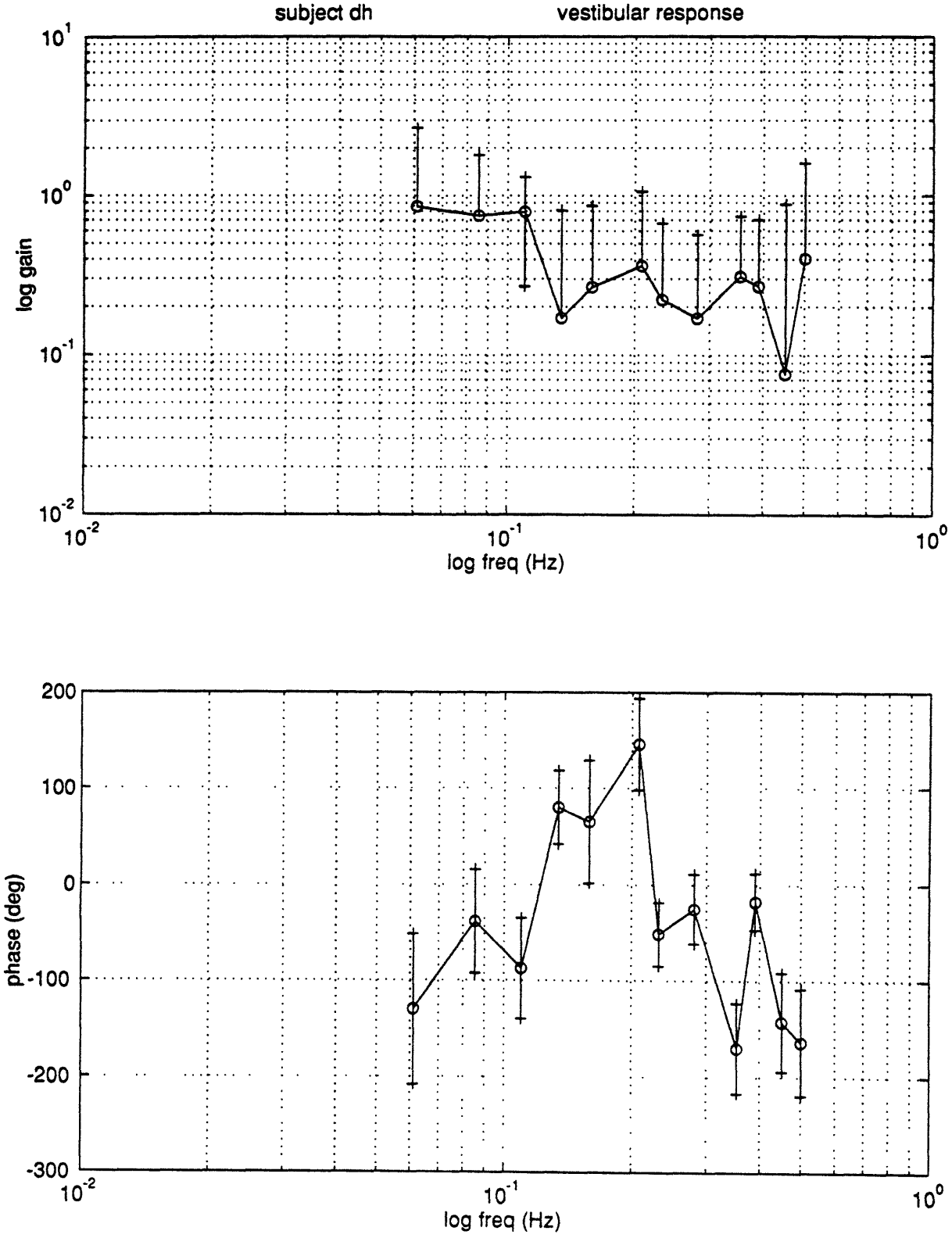

subject dh
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-18

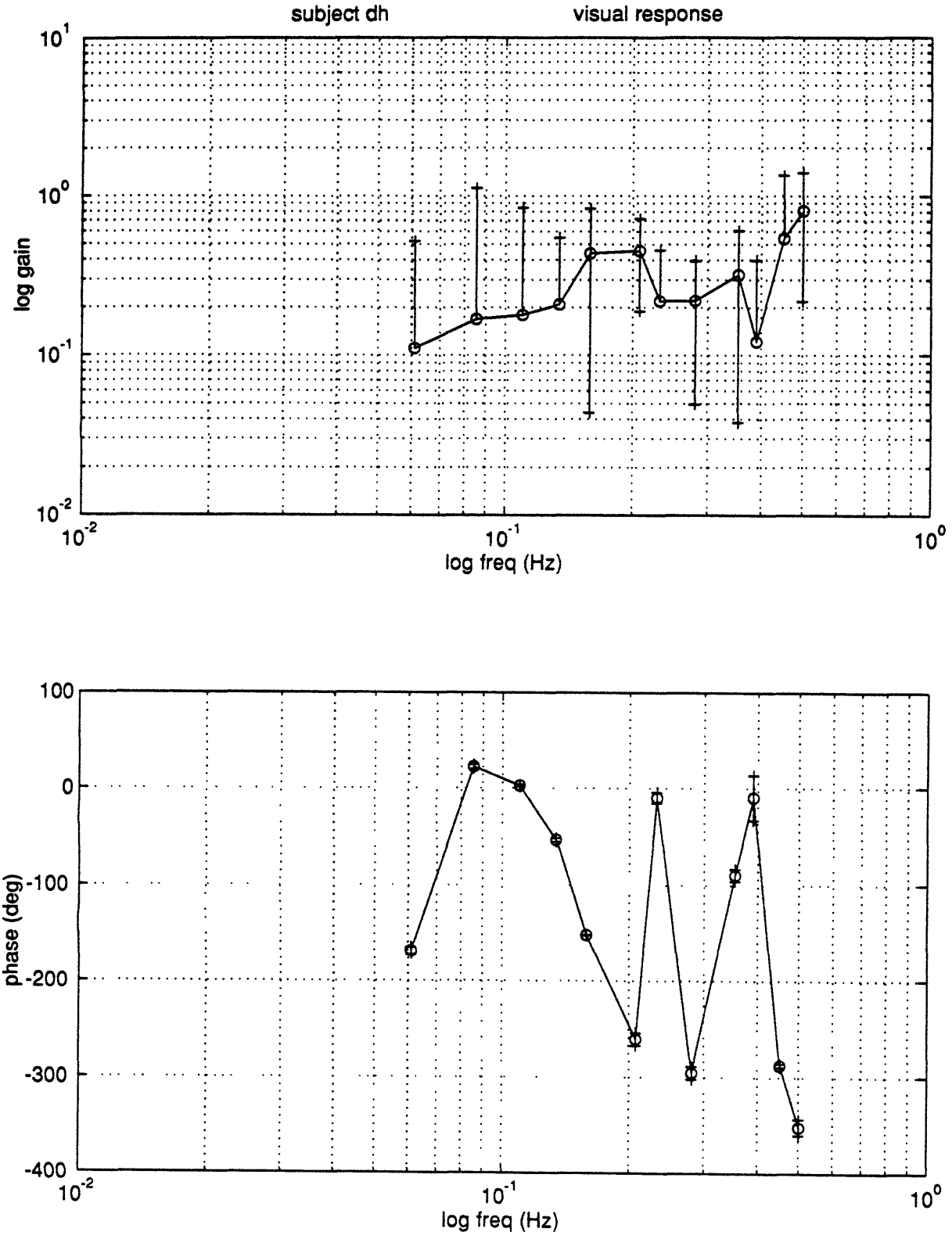

subject dh
visual response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-19

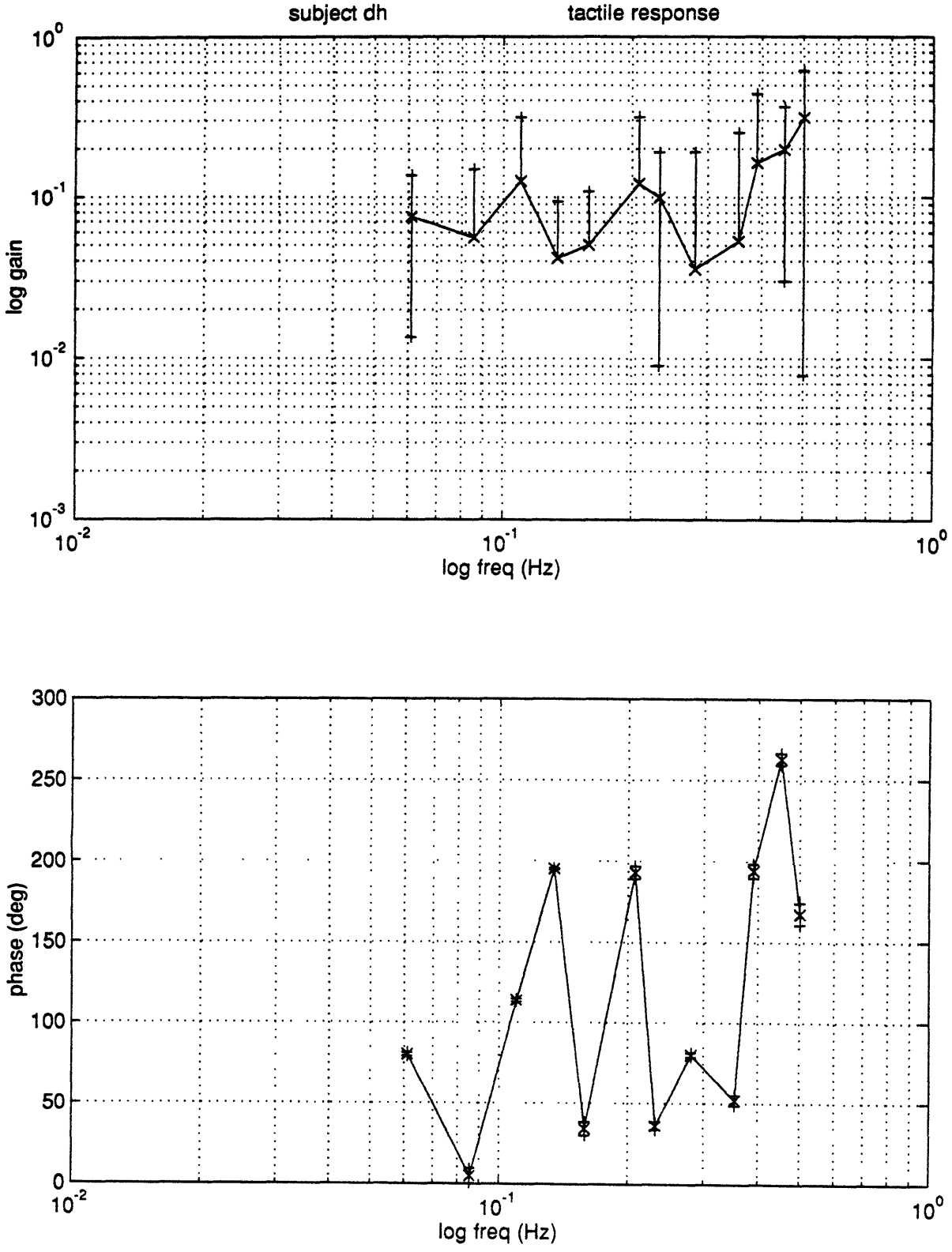

subject dh
tactile response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-20

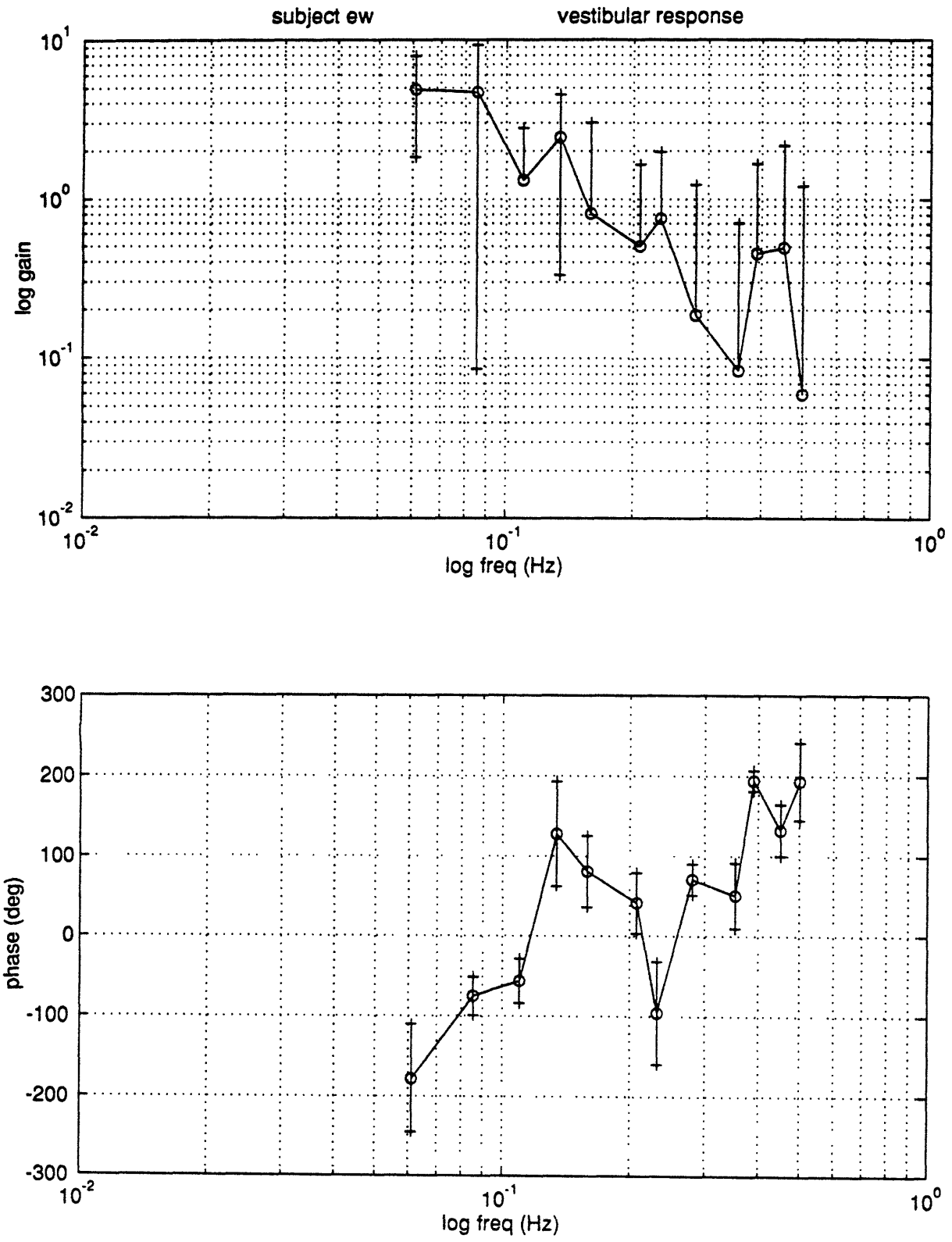

subject ew
vestibular response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-21

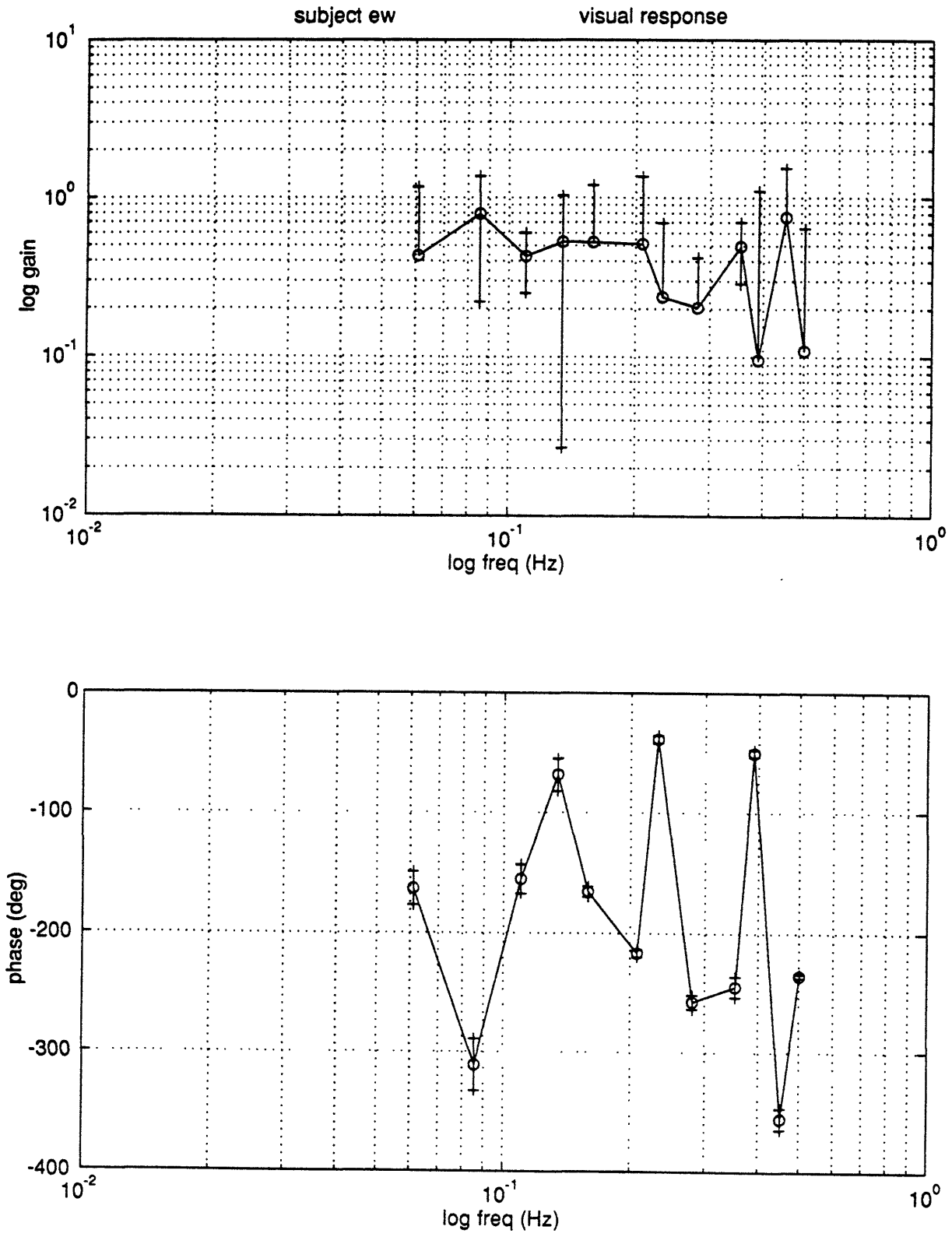
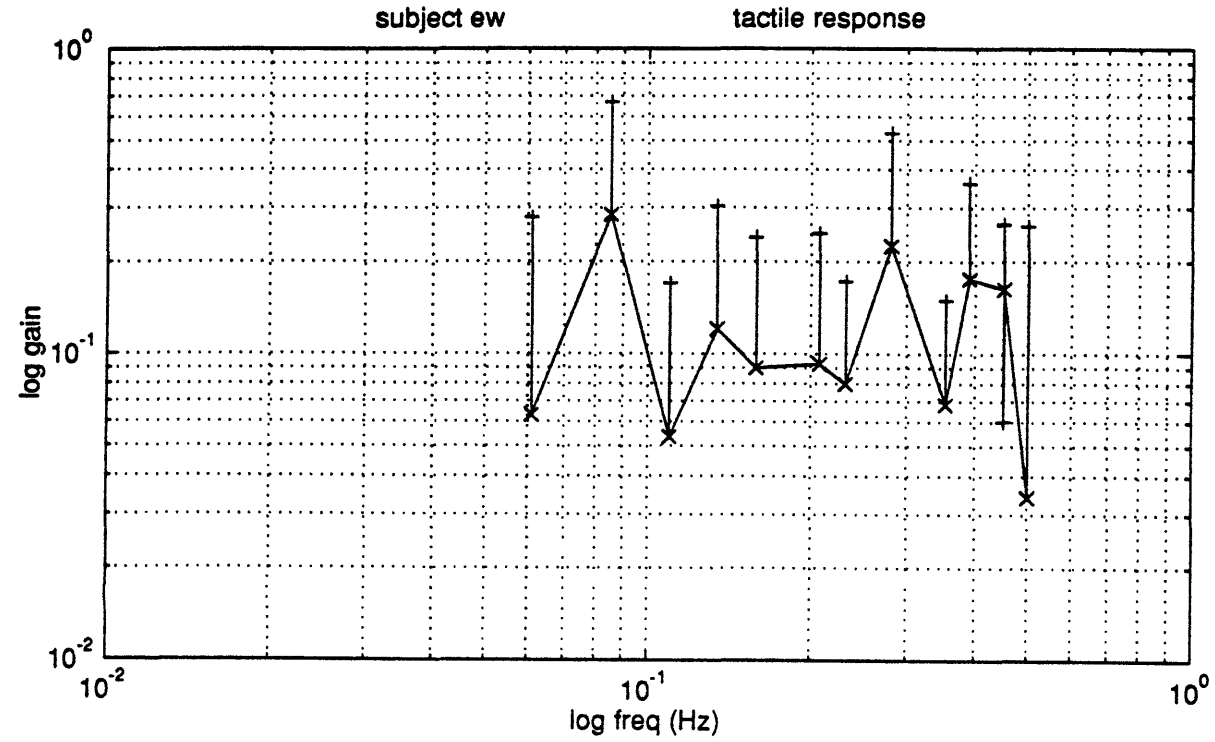
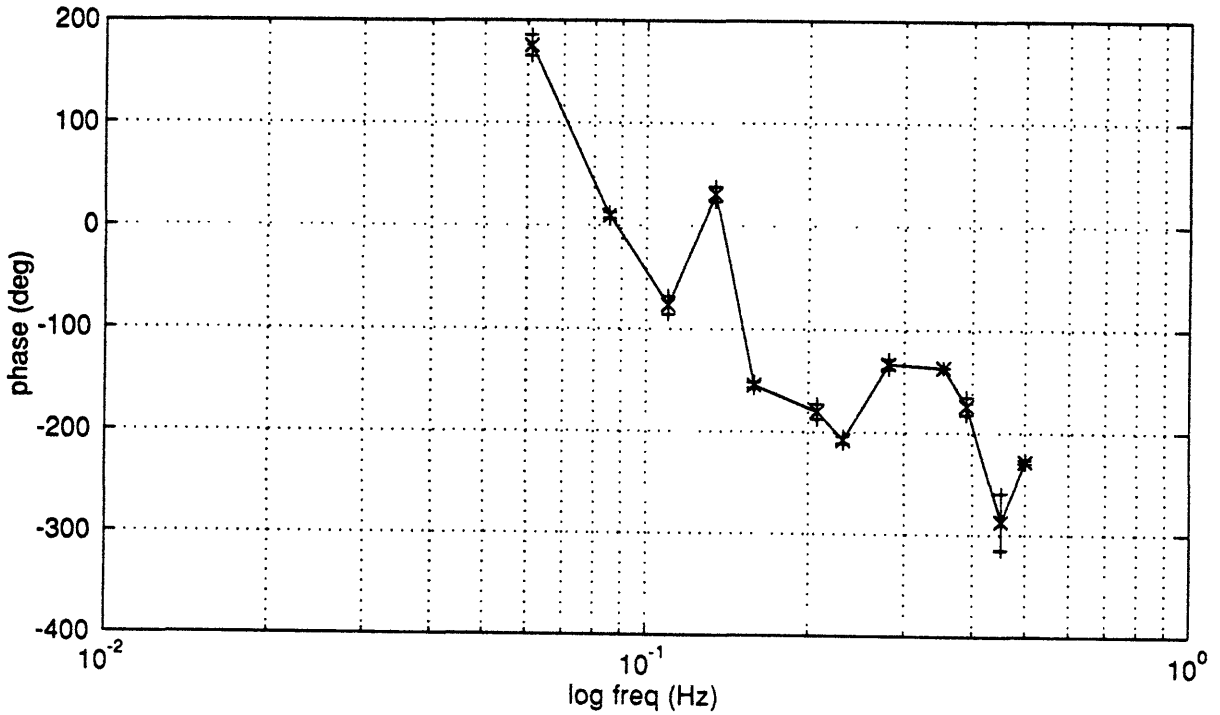

subject ew
visual response
log gain
log freq (Hz)
phase (deg)
log freq (Hz)

Figure 5-22


subject ew
tactile response
log gain
10^0
10^{-1}
10^{-2}
10^{-2}
10^{-1}
10^0
log freq (Hz)


phase (deg)
200
100
0
-100
-200
-300
-400
10^{-2}
10^{-1}
10^0
log freq (Hz)

Time Analysis

In addition to the frequency-based analysis described above, we also performed time-based evaluations of the data. Our purpose in doing so was to discover any trends that might not have been apparent from the Bode plots.

Approach

This analysis was based on the following assumptions. First, that when presented with simultaneous, conflicting stimuli, the subject will attempt to null his perceived motion by relying on whatever sense he feels is most "trustworthy." Second, the subject may have been using cues that were not among the intended three (whole body acceleration, G-seat acceleration cues, and visual velocity). Primarily, we were concerned about vibrations that were proportional to Sled velocity, and possibly unorthodox interpretations of the G-seat cues. Third, during time periods in which some combination of the cues confirm eachother, subjects will be more apt to counter the confirming cues.

Our method of analysis involved tallying the amount of time the subject attempted to null a given stimulus or combination thereof. For example, we examined all the data points during which the Sled was accelerating in the footward direction, and counted those in which the subject was nulling the acceleration by pulling the joystick towards his headward direction. We then flipped signs and obseved how long the subject countered headward acceleration by moving the sick down. Combinations where two or more cues had the same polarity (ie were consistent with each other) were considered confirming cues. Conflicting pairs of cues were also investigated.

We accounted for stimulus detection thresholds by using published results wherever possible. (Melvill Jones and Young (1978)) For visual motion, we used a rate of 1 pixel per second for velocity, and 1 pixel/s^2 for acceleration. The values in the table represent these figures converted into apparent visual motion. We also considered reaction time. Studies have shown that each sensory organ and pathway has a reaction time associated with it (Postman and Egan (1949) and Melvill Jones and Young (1978)). We tried several sets of reaction times, as given Table 5-1.

Potential Cue		Thresholds	Time Delay (reaction time) , s #1	#2	#3
Sled Velocity	(SV)	±0.016 m/s	0.0	0.2	0.0
Acceleration	(SA)	±0.005 g	0.0	0.2	0.6
G-Seat Accel.	(GA)	±0.15 psi	0.0	0.2	0.12
Jerk	(GJ)	±0.15 psi	0.0	0.2	0.12
HMDVelocity	(HV)	±1.5 cm/s	0.0	0.2	0.2
Acceleration	(HA)	±1.5 cm/s^2	0.0	0.2	0.2

Table 5-1 Parameters tested in time-based analysis.

Results

After compiling the data from the six subjects, we grouped their results and performed t-tests to look for significant disturbance effects, using Satterthwaite's Method (Rosner (1995)). Briefly, the test statistic, t is determined by:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where (x_1,s_1) and (x_2,s_2) are the means and variances of two populations to be compared. n_1 and n_2 are the number of samples in each population. Next one must find the number of degrees of freedom d", which is d' rounded down to the nearest integer:

$$d' = \frac{\left(s_1^2/n_1 + s_2^2/n_2\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{(n_1-1)} + \frac{\left(s_2^2/n_2\right)^2}{(n_2-1)}}$$

We performed t-tests between the nulling and additive responses within each sub-matrix of the large matrix described above. Means and variances were obtained by pooling all six subjects. Therefore $n_1=n_2=6$. The null hypothesis is that there is no significant difference between the total number of nulling and additive responses. In the matrix upper half, where cues are considered singly (main diagonal) or in confirming pairs. The t-test results for the sets of reaction times are tabulated below.

#1	SV	SA	GA	GJ	HV	HA
SV	ADD P<0.0005	ADD P<0.0005	ADD P<0.0005	ADD P<0.0005	ADD P<0.0005	ADD P<0.0005
SA	← P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005
GA	← P<0.0005	↑ P<0.0005	0	NULL P<0.025	NULL P<0.005	NULL P<0.005
GJ	← P<0.0005	↑ P<0.0005	0	0	NULL P<0.025	NULL P<0.025
HV	← P<0.0005	↑ P<0.005	0	0	NULL P<0.05	NULL P<0.005
HA	← P<0.0005	↑ P<0.0005	0	0	0	0

Tables 4-3 (1, 2, and 3) Cue interaction P-values for each set of reaction times. Cues are mutually confirming in the matrix upper half. NULL indicates that the subjects predominantly attempted to null out the cue. ADD denotes the opposite, meaning the subjects' responses tended to be in the same direcion as the cue. Below the main diagonal, comparisons are made between conflicting cues. Arrows point toward the cue that the subjects attempted to null a significant majority of the time. Non-significant interactions (or single cues) are marked with zeros.

#2	SV	SA	GA	GJ	HV	HA
SV	ADD P<0.005	NULL P<0.0005	ADD P<0.005	ADD P<0.0005	ADD P<0.005	ADD P<0.005
SA	← P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005
GA	← P<0.0005	↑ P<0.0005	0	0	NULL P<0.005	NULL P<0.005
GJ	← P<0.0005	↑ P<0.0005	0	0	0	0
HV	← P<0.0005	↑ P<0.0005	0	← P<0.05	0	NULL P<0.005
HA	← P<0.0005	↑ P<0.0005	0	← P<0.05	0	0

Table 4-3 (2)

#3	SV	SA	GA	GJ	HV	HA
SV	ADD P<0.05	ADD P<0.025	ADD P<0.025	ADD P<0.05	ADD P<0.05	ADD P<0.05
SA	← P<0.05	0	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005	NULL P<0.0005
GA	← P<0.05	0	0	0	NULL P<0.005	NULL P<0.005
GJ	← P<0.025	0	0	0	0	0
HV	← P<0.025	0	0	0	0	0
HA	← P<0.025	0	0	0	0	0

Table 4-3 (3)

Conclusions

After analyzing the results from all the subjects, examining both frequency content and time history of the responses, we feel this experiment has been a qualified success. Although no major trends emerged from the frequency response curves for the individual subjects, they did reveal some interesting results. Likewise, the time response data, and in particular the t-tests, provide some new insights into how humans rationalize conflicting sensory information. Finally, we feel that significant progress has been made towards the completion of a reliable and flexible integrated system for the presentation of multi-sensory motion cues.

Frequency response

.Several are reprinted here either because they are characteristic of the subject population in general, or because they describe unexpected behavior. Subject JR's response to vestibular stimuli, Figure 6-1, is typical of most subjects. The gain curve shows a clear rolloff, approximately equal to -20 dB / decade. Considering that the input to the vestibular system is an acceleration, and subjects were attempting to null velocity, it follows that somewhere in the combined estimator/ control strategy block, an integration is taking place. This hypothesis is consistent with the data. However, the phase plot shows an excessive amount of variability, especially for the simple dynamics expected from the model. Note that the phase data roughly centers around 180°. Subjects PS, KJ, and MM were qualitatively similar, being roughly out-of-phase. In general, the phase curves shows large swings that were significantly greater than the extents of the error bars. We found this surprising, considering the conservative manner in which we calculated the confidence

intervals. Therefore, the phase data must be viewed with a bit of skepticism. Nevertheless, the presence of the gain rolloff is encouraging.

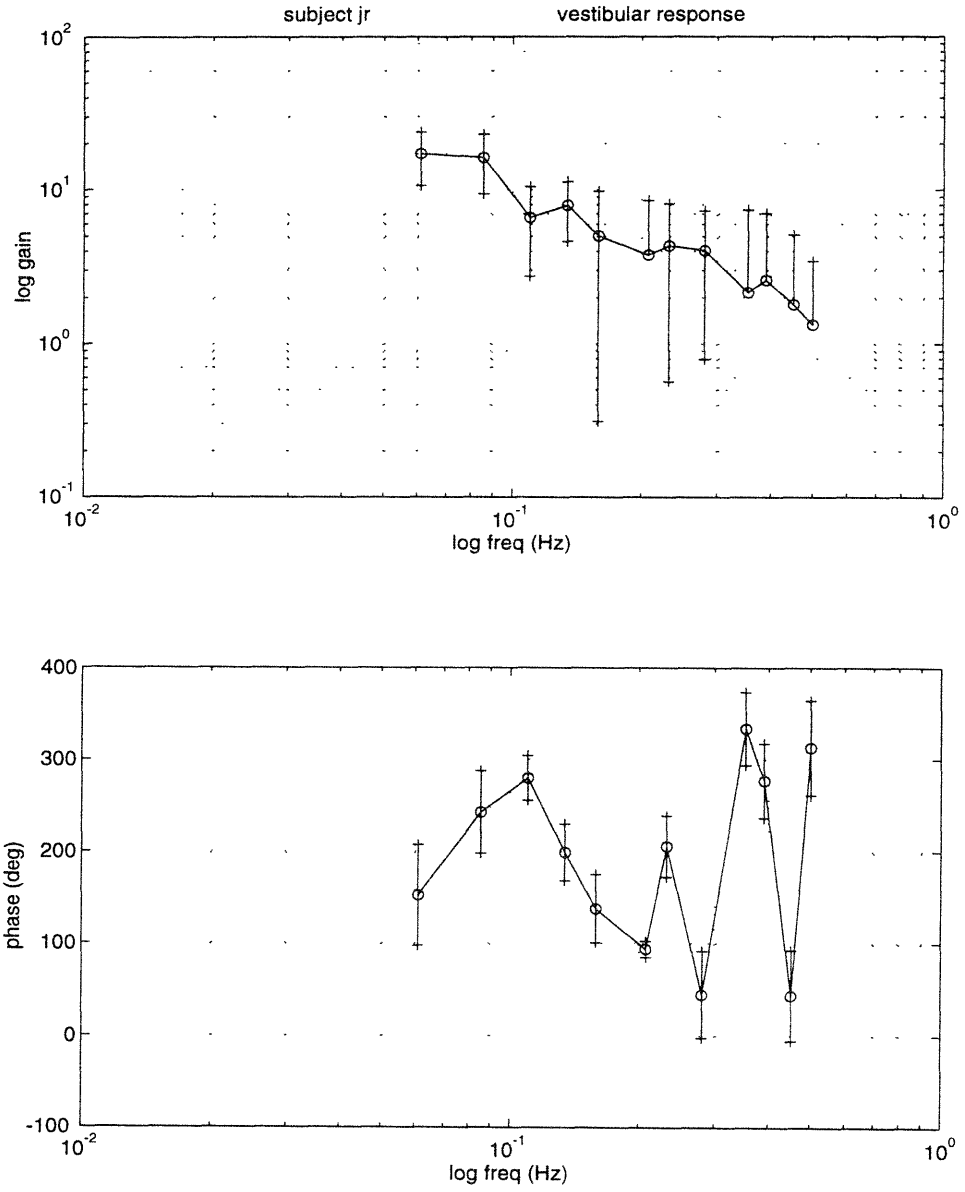


Figure 6-1 Vestibular response for Subject JR.

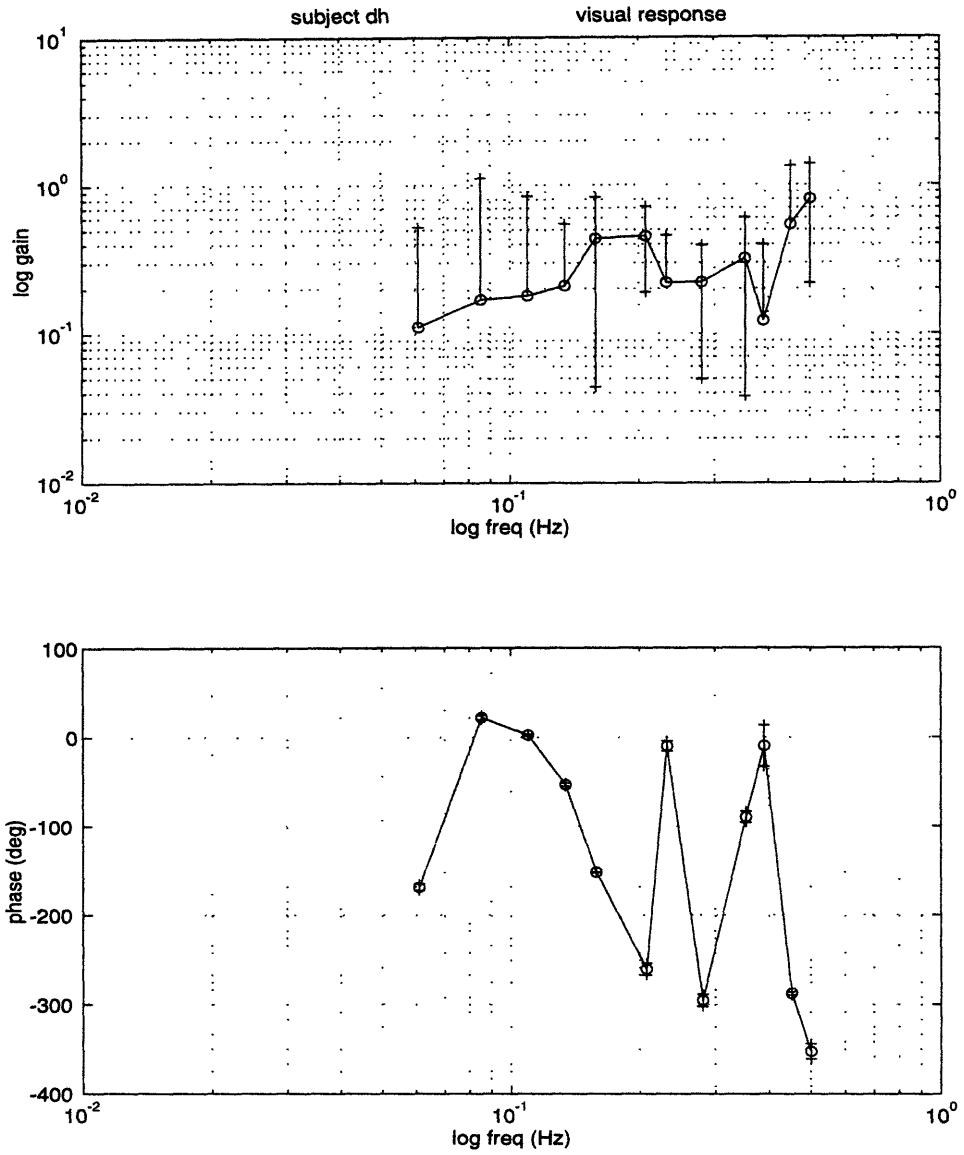


Figure 6-2 Visual response for Subject DH.

In regard to the visual cue, subjects tended to show flat gain curves over the range we tested, or even a possibly a small rise, as Subject DH shows. This goes against the findings of Zacharias (1979) who found significant rolloff in visual gain at the same frequencies. Young (1979) also reported that linearvection gain drops off quickly above 0.1 Hz.

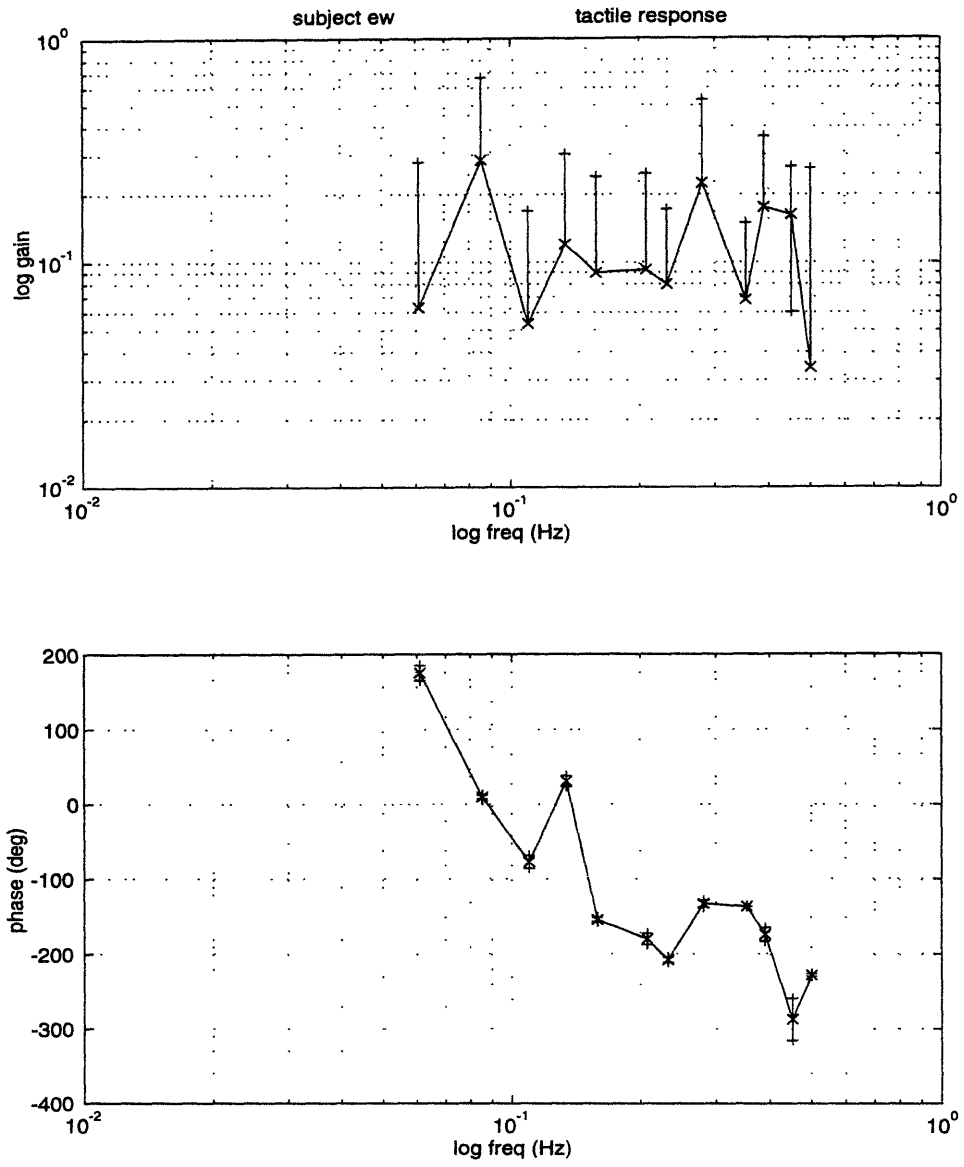


Figure 6-3 Tactile response of Subject EW.

Finally, tactile cues from the G-seat did not produce any consistent results across subjects. Subject EW (Figure 5-4) shows a typically flat but noisy gain curve. The lack of lower bounds on the error bars arises from the method by which they were calculated (see Results Section). The large lead - lag in the phase data is interesting, but no other subjects produced similar results. We speculate this is an artifact from processing.

Using the G-seat with a horizontal (supine) subject raises some serious concerns about its effectiveness. As described in the Equipment section, the G-seat normally acts by changing the pressure distribution, but not the total force, on the occupant's posterior. The subject's weight opposes the pressure in the seat, resulting in a dynamic balance. During horizontal Z-axis trials, the subject's weight no longer acts to counter the seat inflation. As a result, he could be pushed out of the seat by the air bladders, were it not for the safety harness. This is the crux of the problem. As the seat inflates (simulating $-G_z$), the safety belts (especially the lap belts) exert more pressure on the upper thighs. Because the force exerted by the belt is proportional to the pressure in the bladders, the subject is caught in the middle; additional air only serves to increase the true pressure magnitude on the skin, not level out the pressure distribution as intended. The subject may interpret the resulting cue as indicating $+G_z$.

Our solution to the above problem was to adjust the safety harness such that the shoulder belts carried a greater share of the load, and to add an elastic belt tensionner to the lab belt. Together, these steps helped alleviate the "belt squeeze" that was occurring. However, subjective evaluations of the G-seat in this orientation determined that the cue interpretation was still backwards. We decided to adopt this reversed polarity for G-seat cues throughout the analysis.

Time responses

The time data showed that subjects relied heavily on sled acceleration in issuing nulling commands with the joystick. The SA column is interesting, because the elements below the diagonal indicate that when sled acceleration conflicted with other cues, they often chose

to null the sled acceleration. This result is consistent with comments taken during post-session interviews with the subjects, who stated that they discounted the visual and tactile cues once they realized that they could not control those disturbances. The lesson here, for future research, would be to close the control loop on all three inputs.

There is an interpretive problem when one compares an acceleration cue with a velocity command. Assuming that a subject has a limited ability to integrate otolith signals into velocity estimates, then there will be times when his estimates of velocity and acceleration have opposite signs (imagine a sine wave and its derivative, a cosine). At these times, the subject would be adding to his velocity if he were to attempt to null his perceived acceleration. However, by examining interactions where true velocity and acceleration have opposite signs, the subjects null acceleration significantly more often ($P<0.0005$). Rather than accurately integrating, the subjects may be continually rezeroing their perceived velocity based on recent accelerations.

This hypothesis raises an interesting possibility. Because acceleration changes sign twice for each change in velocity, some subjects might command the joystick at twice the frequency of the velocity disturbance. This could give rise to the peaks in the power spectral density that lay beyond the tested frequencies, which are clearly visible in Figure 6-4.

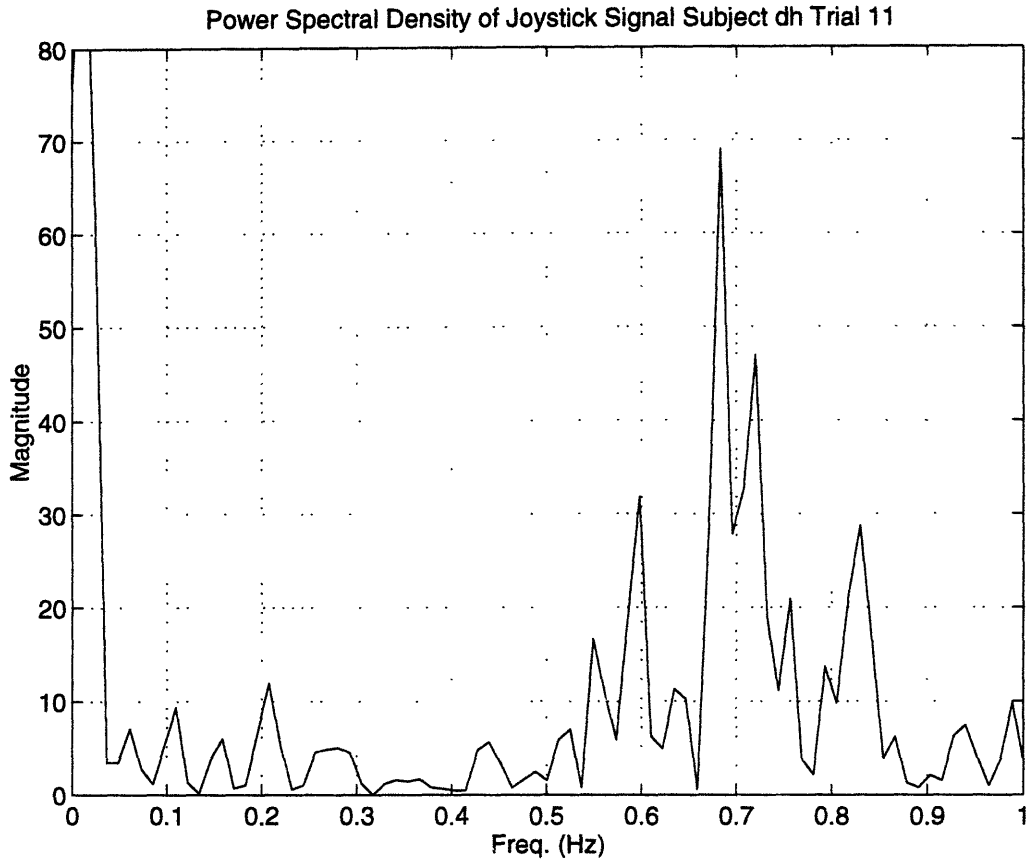


Figure 6-4 Power spectral density of joystick commands for Subject DH during one trial. The high peaks at 0.6, 0.7, and 0.83 Hz lay beyond the tested frequencies. This is evidence of either injected dither or a response to the harmonic of an input frequencies.

The lack of nulling commands for Sled Velocity (SV) suggests that vibration of the cart (which is proportional to velocity) did not greatly aid the subject's performance, although every subject mentioned feeling the vibrations. The presence of additive responses to SV is puzzling, but may be related to the strong acceleration dependence described above.

We were disappointed to find that the G-seat, taken alone, did not produce any significant effects on subject response, with respect to either acceleration or jerk. This can be seen on the main diagonal of the time response matrices (Table 4-3). We believe the problem here lies with the unorthodox orientation of the G-seat with respect to gravity (subject supine),

as described in the previous section. The combination of G-seat as an acceleration cue and the HMD as a velocity cue was significant ($P < 0.005$) in all cases, eliciting correct nulling responses. The same holds when G-seat and visual acceleration cues are confirming ($P<0.005$). Perhaps the utility of these devices for creating motion cues lies in their combined application.

Of the various time delays we applied to the data, neither resulted in as many significant modalities as did the test with zero time delays. Given a larger and more practiced subject population, one could perform a similar experiment and iterate the results on reaction times for each modality. Solving for local maxima in the P values may produce valid estimates of the reaction times for each modality.

Discussion

The crux of this research has been to investigate how interactions between the senses affect the way in which we perceive motion. Therefore, what general conclusions can be made about the dynamics of visual, vestibular, and tactile pathways during simultaneous simulation? Furthermore, what contribution does each make to overall motion perception?

The results of the vestibular portion of the model suggest that subjects were integrating information from their vestibular system to obtain estimates of velocity. This should not come as a surprise, as vestibular integration is a demonstrable effect (Israel et al (1993)). However, the time-response data indicate otherwise, which will be discussed shortly.

Unfortunately, the frequency response data did not reveal any consistent trends from which to draw inferences about tactile or visual influences. The large fluctuations in gain and

phase contraindicate the linear estimator model which we employed. This is not particularly surprising, considering that Zacharias (1979) abandoned a linear model in favor of a non-linear conflict model. The difficulty with our approach was that the frequency response curves presupposed a linear model. Perhaps a non-linear model, or some other form of analysis that does not assume a model, would have produced a more interesting set of data.

We are able to draw some stronger conclusions from the time response data, however. The dominance of the vestibular (acceleration) cues over the other inputs shows how strong a role the vestibular system plays in estimating motion. The high negative correlation between subject responses and sled velocity (i.e. they typically added to the disturbance velocity, rather than nulling it) is curious. Support for this finding lies in the vestibular response phase curves. Several subjects had responses that lagged 180°(a lead of that size is probably an artifact).

If the phase data is credible, then we must grapple with the question of why subjects would perform contrary to their intentions. Perhaps the added distraction of the uncorrelated visual and tactile information resulted in a loss of integration. In this case, the negative correlation is understandable, given the strong positive (nulling) correlation to acceleration, which consistently leads velocity in phase. Subjects may have been responding to instantaneous acceleration, rather than estimating their velocity from it. However, the vestibular gain curves dictate against this hypothesis. Further investigation would be necessary to make a firm conclusion.

The lack of significant trends in the tactile and visual cues does not indicate that these stimuli were ignored by the subjects; otherwise the model would have reduced to a single channel vestibular model, and the remaining vestibular frequency responses would have shown trends more closely resembling those of Young and Meiry (Meiry (1965)). The possibility remains that subjects were influenced by scene and seat motion, but that these influences were small compared with the dominant vestibular cue. This is largely speculation, however. More sophisticated analysis techniques may reveal the presence of a trend.

We were encouraged to find that combined visual and tactile stimuli produced a significant effect on the subject's perception of motion. Further refinements to the G-seat and HMD systems will, hopefully, make these cues more compelling to the subject. We must conclude that, in their current state, these devices cannot provide a sense of motion as compelling as does true whole body movement. However, no foreseeable reason exists why tactile, or visual, cueing devices cannot be refined to a point where they can compete for a person's perception of movement on an equal basis with true motion.

Suggestions for Future Research

Research into multi-sensory perception of motion is a relatively new field. Furthermore, our understanding of the tactile channel is particularly basic. There still much to be determined about how the brain interprets signals from the senses and forms a single conception of movement. Advances in simulator technology have opened new avenues for gaining insight into how we relate to our environment.

The possibilities have been barely tapped for research using G-cueing devices, HMDs, and various forms of vestibular stimuli. Further planned research at the Man-Vehicle Lab will involve other axes of motion, changes in subject tasks, and continued refinements in cue realism. More accurate models of the sensory integration process will be developed and tested as well.

Appendix A Program Code for Virtual Environment

The following programs, written in C/C++, form a portion of the application that runs the visual aspects of the experiment. In use, the program provides a menu-driven user interface, allowing a non-programmer to easily add or modify experiments, maintain a list of subjects (each of whom may follow a separate experimental protocol), and run complete experiment sessions with a minimal workload. Functions provided in World Tool Kit are used extensively to minimize the amount of low-level graphics handling necessary. Please refer to WTK documentation for descriptions of these functions.

The vast majority of the code for this application was written by John DeSouza and Adam Skwersky, both former MIT MS students at the Man-Vehicle Lab. They developed this program for use in circularvection experiments. Only the portions of the code changed for the current research are included. Neither is a full description of the code's function presented here. Please contact Dr. Charles Oman at the MVL for more information.

The author's contribution has consisted mainly of expanding the program for use in linear motion experiments. I have also made additions to the user interface to allow for repeated trials in the event of subject or experimenter error.

Most importantly, I also found a solution the frame rate question, which had been problematic for both the aforementioned researchers and other researchers around the country who use WTK for visual scene rendering. As stated in the text, WTK is free-running, meaning that there is no provision for setting or controlling the frame rate. The resultant fluctuations in scene velocity can destroy a subject's state of vection. The simple solution involved attaching a function generator to the PC's serial port, and a segment of new code that instructs the computer to wait for a rising pulse from the FG before rendering the next frame. The only caveat is that the FG must be set at a frequency below the slowest rate at which the program executes in free-running mode. Otherwise, cycles may be skipped. In use, we achieved an accuracy of 99.96%. Running nominally at 25 fps, at the end of an 81.92 second trial, the scene was out of synchronization with the other devices by only a few hundredths of a second.

A short segment of program was also added that allowed synchronized starts of the visual scene and other equipment. This also relied on a signal to the serial port, in this case from the Sled computer. **New code is given in bold type**. All compiling was done using MetaWare High C/C++, from MetaWare Inc., Santa Cruz, CA.

```
 checkcom.c   /* This short program is NOT part of the application. It is used to
check the
                    status of the computer's serial (COM) port

/* example program to check status of serial port. High C
  Library Reference p. 91 */

#include<stdio.h>
#include<bios.h>
#define _COM2  1

void main() {
        int i;
        int ring, change;
        unsigned data;          /* Output byte */
        unsigned status;        /*Return value */
        char bits[18];          /*Status bit pattern */
        static int pow2[16] =
                {1, 2, 4, 8, 16, 32, 64, 128, 258, 512,
                        1024, 2048, 4096, 8192, 16384, 32768};
        for (int count = 1; count <= 2000; count++) {
         status = _bios_serialcom(_COM_STATUS,_COM2,data);
         for (i = 0; i <= 15; i++) {
                if ((status & pow2[15 - i]) != 0)
                        bits[i] = '1';
                else
                        bits[i] = '0';
         }
         bits[16] = 0;
         if ((status & 64) != 0) ring = 1;
         else ring = 0;
         if ((status & 4) !=0) change =1;
         else change =0;
         /*printf("Status bit pattern of COM1:%s  %d \n",bits,ring);*/
         /*if ((status & 64) != 0) printf("1");
         else printf("0");*/
         printf(" %d  %d %s \n",ring,change,bits);
 }
}
```

```
move.c
#include "wt.h"
#include "move.h"
#include "fplan.h"
#include<time.h>

int ROTATE_VIEWPOINT;

float velocity;          /*angular velocity in rad/sec          */
int is_accelerating;     /*is it constant accelerating (not jerk)*/
float acceleration;      /*angular_accel in rad/sec/sec         */
float *linear_path;      /*array of points for viewpt motion    */
extern int fplan_linear_num_pts; /* # of data points in motion  */
int lin_next;            /*index into linear path               */
int is_linear;           /*flag for linear motion               */
extern int _COM2;
extern int SERIAL_PORT_FPS_FLAG;
unsigned serial_port_status;
int clock_flag;
WTp3 orig_position;/*viewpt position at beginning of motion*/
WTp3 position;           /*current viewpt position              */
int is_jerking;          /*is it const jerking? d(accel)/dt != 0 */
float jerk;              /*angular jerk in rad/sec/sec/sec      */

int num_cycles;          /*number of cycle in the 'path'*/
int cur_cycle;           /*current point in the path      */

int rot_count;
/*wont use angle_path*/
float *angle_path=NULL;  /*an array of angles we want  */

float view_angle=0.0; /*the current rotation angle (might not be actual viewing angle with
head tracker on)*/
float rotate_angle;      /*the amount (rads) to rotate view/object in a cycle*/
float rotationdirection=1.0;
WTobject *itsObject=NULL;/*the room object in case we need it*/

extern int noisy;

void move_init(void){
        view_angle=0.0;
        rot_count=0;
/*      move_setrotdirection(DIRECTION_CW);*/
        move_reset();
}

int move_isdone(void){
        return (cur_cycle>num_cycles);
```

```
}

float move_getangle(void){
        return (view_angle);
}

int move_getrot_count(void){
        return rot_count;
}

void move_setobject(WTobject *anobject){
        itsObject=anobject;
}
void move_linear(void){
        position[Y] = orig_position[Y] + linear_path[lin_next];
        lin_next++;
}
void move_accelerate(void){
        rotate_angle = acceleration/(2*EXPECTED_FPS*EXPECTED_FPS) +
velocity/EXPECTED_FPS;
        velocity += acceleration/(EXPECTED_FPS);
}
void move_jerk(void){
      rotate_angle = jerk/(6*EXPECTED_FPS*EXPECTED_FPS*EXPECTED_FPS)+
              acceleration/(2*EXPECTED_FPS*EXPECTED_FPS) +
velocity/EXPECTED_FPS;
        velocity += acceleration/(EXPECTED_FPS) +
jerk/(2*EXPECTED_FPS*EXPECTED_FPS);
        acceleration += jerk/(EXPECTED_FPS);
}

void move_domove(void){
        if (SERIAL_PORT_FPS_FLAG) {
                clock_flag = move_serial_port_clock(clock_flag);
        }
        if (is_linear) {
                move_linear();
                /*_dos_gettime(&lin_cur_time);
                printf("dos time:
%d:%d\n",lin_cur_time.second,lin_cur_time.hsecond);*/
                /*printf("clock time: %ld \n",clock());*/
                WTviewpoint_setposition(WTuniverse_getviewpoint(),position);
        }
        else {
                if (is_accelerating)
                move_accelerate();
                else if (is_jerking)
                move_jerk();
```

```
                view_angle += rotate_angle;

        /*WTobject_rotate(object,rotate_axis,rotate_angle,WTFRAME_WORLD);*/

        WTviewpoint_rotate(WTuniverse_getviewpoint(),Z,rotationdirection*rotate_angle,
WTFRAME_VPOINT);
                rot_count++;
        }
}
int     move_serial_port_clock(int clock_flag){
        int clock_bit;
        int data;       /* just a place holder */
        do {
                serial_port_status =
_bios_serialcom(_COM_STATUS,_COM2,data);
                if (serial_port_status & 32) { /* Operator stop command on DSR pin
6 */
                        printf("Operator commanded Stop");
                        WTuniverse_stop();
                }
                clock_bit = (serial_port_status & 16); /*CTS pin 8 on DB-9*/
                if (!clock_bit) clock_flag = 0;
        } while (!clock_bit || clock_flag);
        clock_flag =1;
        return clock_flag;
}


void    move_reset(void){
        is_jerking=is_accelerating=0;
        is_linear=lin_next=0;
        clock_flag =0;
        acceleration=jerk=0.0;
        if (angle_path){
                free(angle_path);
                angle_path=NULL;
        }
        cur_cycle=num_cycles=0;
}

void    move_move_viewpoint(void){
        is_linear=1;
        lin_next=0;
        WTviewpoint_getposition(WTuniverse_getviewpoint(),orig_position);
        position[Y] = orig_position[Y];
}

void    move_const_velocity(float veloc,float time){
        /*velocity in ~degs/sec*/
```

```
        /*WTK wants it in rads*/
        if (noisy)
                printf("move_const_velocity: veloc=%f time=%f",veloc,time);
        move_reset();
        rotate_angle = veloc * PI/180 / EXPECTED_FPS;
        velocity = veloc * PI/180;
        num_cycles=(int) EXPECTED_FPS*time;
        if (noisy) printf("acceleration = %f, num_cycles = %d, velocity = %f, rotate_angle
= %f",
                acceleration,num_cycles,velocity, rotate_angle);
}

void    move_const_accel(float V0, float accel,float time){
        /*accelertion in deg/sec/sec, convert to rad/sec/sec      */
        /*V0 initial velocity in deg/sec                          */
        if (noisy) printf("move_const_accel: V0=%f accel=%f time=%f\n",V0,accel,time);
        move_reset();
        acceleration = accel*PI/180;
        is_accelerating = 1;
        num_cycles= time*EXPECTED_FPS;
        velocity = V0 * PI/180;
        rotate_angle = velocity / EXPECTED_FPS;
        if (noisy) printf("acceleration = %f, num_cycles = %d, velocity = %f, rotate_angle
= %f",
                acceleration,num_cycles,velocity, rotate_angle);
}

void    move_const_jerk(float A0,float V0, float jk, float time){
        /*A0, V0 initial accel and veloc in deg           */
        /*jk is jerk in deg/sec/sec/sec             */
        move_reset();
        jerk = jk*PI/180;
        is_jerking = 1;
        num_cycles = time/EXPECTED_FPS;
        velocity = V0 * PI/180;
        rotate_angle = velocity / EXPECTED_FPS;
        acceleration = A0 * PI/180;
}

void    move_getkinetics(float *v,float *a,float *j){
        *v=velocity;
        *a=acceleration;
        *j=jerk;
}

void    move_setrotdirection(int rotdir){
        rotationdirection = (rotdir==DIRECTION_CW)?(1.0):(-1.0);
        printf("Setting rotation direction to %f\n",rotationdirection);
}
```

fplan.c

```
#include <stdio.h>
#include "fplan.h"
#include "move.h"
#include "myprint.h"

movement_rec *FlightPlan=NULL;   /*array of movements*/
int          fplan_num_movements;      /*number of movements in array*/
int          fplan_linear_num_pts;
int          fplan_index;          /*an index into the FlightPlan. Used for two
purposes*/
FILE         *fplan_file=NULL;     /*file used to load from file*/
int          all_done;
int          fplan_actual_updates;
float        EXPECTED_FPS;
extern       int          noisy;
extern       float        *linear_path;
/*enum system is as follows: two digit number
  first digit is which is constant, the second
  is which parameter specifies the end of the motion.
  disp,veloc, accel, jerk, time = 0, 1, 2, 3, 4 respectively
  numbers greater 100 are for linear motion
*/

enum {
        CONST_VELOC_TIME = 14,
        CONST_ACCEL_DISP = 20,
        CONST_ACCEL_TIME = 24,
        LINEAR_MOTION = 100

};

void    fplan_load(char *name){
        fplan_index = 0;
        fplan_actual_updates = 0;
        all_done=0;
        move_init();
        printf("trying to open %s...",name);
        if (fplan_file = fopen(name,"r")){
                int cnt;
                fscanf(fplan_file,"%d",&fplan_num_movements);
                printf("%d movements.\n",GetNM());
                FlightPlan = (movement_rec *) malloc(sizeof(movement_rec)*GetNM());
                for (cnt=0;cnt<fplan_num_movements;cnt++)
                        fplan_read_next_movement();
        }
        else
                printf("failed!\n");
```

```
        if (fplan_file) fclose(fplan_file);
        fplan_file=NULL;
        fplan_index=-1;
        fplan_start_next_movement();
}

void fplan_picknload(void) {
        fplan_load("fplan.txt");
}

void fplan_read_params(int numparams){
        int cnt;
        if (noisy) printf("parameters: ");
        for (cnt=0;cnt<numparams;cnt++) {
                fscanf(fplan_file,"%f",&(FP(fplan_index).p[cnt]));
                if (noisy) printf("%f\t",FP(fplan_index).p[cnt]);
        }
        if (noisy) printf("\n");
        /*zero the rest*/
        while (cnt<FPLAN_MAX_PARAMS)
                FP(fplan_index).p[cnt++]=0.0;

}
void fplan_read_path(){
        int cnt;
        fscanf(fplan_file,"%d",&fplan_linear_num_pts);
        linear_path = malloc(sizeof(float) * fplan_linear_num_pts);
        if (linear_path == NULL) printf("Memory allocation error");
        for (cnt=0;cnt<fplan_linear_num_pts;cnt++) {
                fscanf(fplan_file,"%f",&(linear_path[cnt]));
                if (noisy) printf("# %d : %f\t ",cnt,linear_path[cnt]);
        }
        FP(fplan_index).NumUps = fplan_linear_num_pts;
}
void fplan_read_next_movement(void){
        fscanf(fplan_file,"%d",&(FP(fplan_index).type));
        if (noisy)
                printf("movement %d of type %d\n",fplan_index,FP(fplan_index).type);

        switch (FP(fplan_index).type) {
          case CONST_VELOC_TIME:
                fplan_read_params(2);
                FP(fplan_index).NumUps = EXPECTED_FPS*FP(fplan_index).p[1]+1;
                break;
          case CONST_ACCEL_DISP:
                fplan_read_params(3);
                break;
          case CONST_ACCEL_TIME:
                fplan_read_params(2);
```

```
            FP(fplan_index).NumUps = EXPECTED_FPS*FP(fplan_index).p[1]+1;
            break;
        case LINEAR_MOTION:
            fplan_read_path();
            if (noisy) printf("in fplanreadnextmovement case
LINEAR_MOTION \n");
            break;
        default:
            /*oops, unsupported movement*/
            break;
    }
    fplan_index++;
}

void    fplan_start_next_movement(void){
    float v,a,j;
    fplan_index++;
    switch(FP(fplan_index).type){
        case CONST_VELOC_TIME:
            move_const_velocity(FP(fplan_index).p[0],FP(fplan_index).p[1]);
            break;
        case CONST_ACCEL_DISP:
            /*assume V0=0 for now*/
            move_getkinetics(&v,&a,&j);
            move_const_accel(v*180.0/PI,FP(fplan_index).p[0],
                        sqrt(2*FP(fplan_index).p[1]/
                            FP(fplan_index).p[0]));
            break;
        case CONST_ACCEL_TIME:
            move_getkinetics(&v,&a,&j);
            move_const_accel(v*180.0/PI,FP(fplan_index).p[0],
                        FP(fplan_index).p[1]);
            break;
        case LINEAR_MOTION:
            move_move_viewpoint();
            break;
        default:
            /*not supported, dont do anything*/
            break;
    }
}

void    fplan_exit(void){
    printf("in fplan_exit...\n");
    /*fplan_file is usually closed in fplan_load*/
    if (fplan_file)
        fclose(fplan_file);
    fplan_file=NULL;
    if (FlightPlan)
```

```
                free(FlightPlan);
        FlightPlan=NULL;
}

int     fplan_expups(void){
        /*number of expected updates*/
        int cnt;
        int expups=0;
        for (cnt=0;cnt<fplan_num_movements;cnt++)
                expups+=FP(cnt).NumUps;
        return (expups);
}

void fplan_domove(void){
        if (FP(fplan_index).NumUps==0) {
                if ((fplan_index+1)<fplan_num_movements)
                        fplan_start_next_movement();
                else (all_done=1);
        }
        fplan_actual_updates++;
        move_domove();
        FP(fplan_index).NumUps--;
        myprint("%d ",FP(fplan_index).NumUps);
}

void    fplan_setexpectedfps(float efps){
        EXPECTED_FPS = efps;
}
```

runexp.c

```
/*runexp.c*/

#include "wt.h"
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <bios.h>

#include "runexp.h"
#include "joysens.h"
#include "spea2d.h"
#include <time.h>
#include "datarec.h"
#include "fplan.h"
#include "move.h"
#include "myprint.h"
#define NUMUPS     886

/*#define CHANGEORDER 1*/
void spin(WTobject *object);
void SetupSpin(int howmany);
void CleanupSpin(void);
static void display_menu();
static void setparameters();
void draw2d(short eye);
void toggle_joystick();
void record_joystick();
FILE *infile = NULL;
FILE *outfile = NULL;

int num_rotation, num_tests, num_rotupdate;
char rotate_axis = Y;


WTpq lobbyview;
extern WTpq *spinView;
WTobject *lobby = NULL;
extern WTobject *spinObject;
extern int noisy;
/* frame rate checker */
static clock_t currtime;
static clock_t prevtime;
extern char     SceneDir[80];
extern char     SubjectDir[80];
extern char     ProfileDir[80];
int SERIAL_PORT_TRIGGER = 1;      /* set to 0 for no external trigger */
int SERIAL_PORT_FPS_FLAG = 1;     /* set to 0 for free running */
```

```
int _COM2 = 1;
/*joystic stuff*/

void benchit();
void (*actionfn) ();
void WTuniverse_setactions(actionfn);
void datastamp();
TreatmentInfoRec TInfo;

void runexp(TreatmentInfoRec *Info, char *subjectname)
{

        WTp3 centre;
        FILE *infile = NULL;
        int rend;
        short myevents[3];
        float fr,totime;
        char temp[90];
        myevents[1]=WTEVENT_ACTIONS;
        myevents[0]=WTEVENT_OBJECTSENSOR;
        myevents[2]=WTEVENT_TASKS;



        if (Info) TInfo = *Info;
        else return;

        printf("Fplanfile is %s\t",TInfo.fplanfile);
        printf("scenefile is %s\n",TInfo.scenefile);
        strcpy(temp,TInfo.triggdir);
        strcat(temp,subjectname);
        temp[strlen(temp)+1]=0;
        temp[strlen(temp)]='\\';
        strcat(temp,TInfo.triggfile);
        strcpy(TInfo.triggfile,temp);
        printf("Trigg file is %s\n",TInfo.triggfile);
        printf("Treatment Code is %s\n",TInfo.treatcode);
        printf("Expected FPS is %f\n",TInfo.expected_fps);
#ifdef DIRECTION_TOO
        printf("Direction is %s\n", (TInfo.direction==DIRECTION_CW)?"CW
":"CCW");
        move_setrotdirection(TInfo.direction);
#endif
        /*Initialize the universe, name it rotate, and set background to black*/
        /*WTdraw_setactions(draw2d);*/
        /*WTuniverse_new(WTDISPLAY_STEREO, WTWINDOW_DEFAULT);*/
        WTuniverse_setname("rotate");
        WTuniverse_setactions(benchit);
```

```
#ifdef CHANGEORDER
        if (WTuniverse_seteventorder(3,myevents))
                printf("changed order\n");
#endif

        WTuniverse_setbgcolor(0xFFF);

        WTuniverse_setresolution(WTRESOLUTION_LOWNTSC);
        /* Load the room as an object and assign the spin procedure to it.
          Set viewpoint to that loaded from file*/

        WTpq_init(&lobbyview);

        lobby = WTobject_new(TInfo.scenefile,&lobbyview,1.0,TRUE,TRUE);
        if (lobby) printf("Loaded the scene in/\n");
        spinObject = lobby;
        spinView = &lobbyview;
        /*printf("Number of polygons: %d\n",WTuniverse_npolygons());*/
        /*printf("Number of polygons in object %d\n",WTobject_npolygons(lobby));*/
        WTviewpoint_moveto(WTuniverse_getviewpoint(), &lobbyview);
        WTobject_settask(lobby,spin);

        /*Set the pivot point as the centre of the room*/
        WTuniverse_getmidpoint(centre);
        WTobject_setpivot(lobby,centre,WTFRAME_WORLD);

        /* Set default stereo viewing paramenters */
        WTviewpoint_setconvergence(WTuniverse_getviewpoint(),-63);
        WTviewpoint_setparallax(WTuniverse_getviewpoint(),2.034);
        WTviewpoint_setaspect(WTuniverse_getviewpoint(),0.81);
        WTviewpoint_setviewangle(WTuniverse_getviewpoint(),0.655);

        /* Set rendering for textured with perspective
          correction and gouraud shading. */
        rend = WTuniverse_getrendering();
        /*printf("rendering is %ld\n",rend);*/
        rend = rend | WTRENDER_TEXTURED | WTRENDER_PERSPECTIVE;
        if (noisy){
                printf("rendering is %ld\n",rend);
                printf("shading is %ld\n",WTRENDER_SHADED);
        }
        WTuniverse_setrendering(WTRENDER_V_TEXTURE |
WTRENDER_PERSPECTIVE );
        if (noisy && 0){
                printf("Parallax is set at:%f.\n",
                    WTviewpoint_getparallax(WTuniverse_getviewpoint()));
                printf("Converegence is set at: %d.\n",
                    WTviewpoint_getconvergence(WTuniverse_getviewpoint()));
        }
```

```
/* load lights from file */
WTlight_load("mylights");

/*Set ambient light */
WTlight_setambient(0.46);

/* open the keyboard */
WTkeyboard_open();



num_tests=1;

if (!datarec_openexp(TInfo.triggfile,NUMUPS))
        printf("Problem opening the daq module");

else printf("DAQ open and waiting for data\n");

WTobject_remove(lobby);
/* Perform the test */


setparameters();
WTobject_add(lobby);
WTuniverse_ready();
if (SERIAL_PORT_TRIGGER) {  /*this will pause until a RING is */
        WTuniverse_go1();  /*detected at COM2, then resume   */
        unsigned data, status; /*on DB-9 cable: 9=ring, 5=sig grnd */
        do {
                status = _bios_serialcom(_COM_STATUS,_COM2,data);
        } while ((status & 64) == 0);
}
toggle_joystick();
prevtime = clock ();
/*initial data stamp*/
datastamp(data_rec_trigger1);
WTuniverse_go();
currtime = clock ();
/*one last datastamp*/
datastamp(data_rec_trigger1);
WTuniverse_setbgcolor(0x000);
WTuniverse_go1();
num_rotupdate = fplan_get_actualupdates();
fplan_exit();
datarec_closeexp();
totime=((float) currtime-prevtime)/((float) CLOCKS_PER_SEC);
fr=num_rotupdate/totime;
printf("clock calls : prev: %ld current: %d", prevtime, currtime);
printf("Total time: %5.2f\tframe rate %5.2f\n", totime,fr);
```

```
        CleanupSpin();

        printf("Vacuuming the universe...swooooooop....");
        toggle_joystick();
        /* all done; clean everything up.*/
        WTuniverse_vacuum();
        printf("Turning off the LIGHTS!\n");
        WTlight_deleteall();
        WTlight_setambient(0.0);
        printf("<slurp> <slurp>\n");
}


/***********************************************
      Set parameters for rotation
***********************************************/
static void setparameters()
{
        int num=13;

        /*NUMBER OF ROTATIONS*/
        /*ROTATION SPEED*/
        /*rotate_angle = ((float) num)*PI/1440.0;      */
        /*num_rotupdate = round((2*PI*((float)num_rotation))/(rotate_angle));*/

        fplan_setexpectedfps(TInfo.expected_fps);
        fplan_load(TInfo.fplanfile);

        num_rotupdate=fplan_expups();
        printf("Number of expected rotation updates: %d\n",num_rotupdate);
        SetupSpin(num_rotupdate+GetNM()*5);

}

/***********************************************
         Display option menu
***********************************************/

void display_menu()
{
        printf("    'q' Quit:\n");
        printf("    '?' Print this info:\n");
        printf("    'k' Increase parallax:\n");
        printf("    'l' Decrease parallax:\n");
        printf("    'a' Increase convergence:\n");
        printf("    's' Decrease convergence:\n");
}
```

expmod.c

```
/*Experiment Module: Entry point into runexp*/
#include "wt.h"
#include<stdio.h>
#include "runexp.h"
#include "expmod.h"
#include "move.h"
#include <direct.h>
#include "myprint.h"
#define NUM_TREATS 100
/*#define INDEPENDENT 1*/
int noisy = 0;

void loadexperimentinfo(void);
void expmod_readtreatment(FILE *tfile,TreatmentInfoRec *Treat);
void doexperimentmenu(void);
void load_configuration(char *config);
void makeifnecessary(char *name);
void donexttreatment(void);

TreatmentInfoRec Treatments[NUM_TREATS];
static int      NumTreats;
static char     ExpFileName[80];
static char     SceneDir[80];
extern char     SubjectDir[80];
static char     ProfileDir[80];
static  char    cwd[100];
static  char    itsConfigFile[1024];
extern int      SERIAL_PORT_TRIGGER;
void expmod_setconfigfile(char *afile){
        strcpy(itsConfigFile,afile);
        load_configuration(afile);
        loadexperimentinfo();
}
int expmod_getnumtreats(){
        return NumTreats;
}

SubjectInfo itsSInfo;
#ifdef INDEPENDENT
void main(int argc, char *argv[]){

#else
void expmod_main(SubjectInfo *SInfo){
        if (SInfo) itsSInfo = *SInfo;
        else return;
#endif
/*load configuration*/
```

```
#ifdef INDEPENDENT
        load_configuration("vectexp.cfg");
#else
        load_configuration(itsConfigFile);
#endif

#ifdef INDEPENDENT
        WTuniverse_new(WTDISPLAY_STEREO, WTWINDOW_DEFAULT);
#endif
    loadexperimentinfo();

/*Do the treatment*/
#ifdef  INDPENDENT
    doexperimentmenu();
#else
        donexttreatment();
#endif

/*cleanup*/
#ifdef INDEPENDENT
        WTuniverse_delete();
#endif

}

void    makeifnecessary(char *name){
        char    testdir[120];
        strcpy(testdir,cwd);            /*copy wd into a string*/
        strcat(testdir,name);   /*append the dirname to check*/
        testdir[strlen(testdir)-1]=0;   /*remove the last '\'*/
        mkdir(testdir);                 /*make it if need to*/
}

void expmod_readtreatment(FILE *tfile,TreatmentInfoRec *Treat){

        strcpy(Treat->fplanfile,ProfileDir);
        fscanf(tfile,"%s",Treat->fplanfile + strlen(ProfileDir) );
        myprint("fplan file is in %s\n",Treat->fplanfile);
        strcpy(Treat->scenefile,SceneDir);
        fscanf(tfile,"%s",Treat->scenefile + strlen(SceneDir));
        strcpy(Treat->triggdir,SubjectDir);
        fscanf(tfile,"%s",Treat->triggfile);
        fscanf(tfile,"%s",Treat->treatcode);
        fscanf(tfile,"%f",&Treat->expected_fps);

#ifdef DIRECTION_TOO
        fscanf(tfile,"%d",&Treat->direction);
#endif
```

```
        fscanf(tfile,"%d",&Treat->posture);
        /*Create the directories if they dont already exist*/

        makeifnecessary(ProfileDir);
        makeifnecessary(SceneDir);
        makeifnecessary(SubjectDir);
}

void loadexperimentinfo(void){
        FILE *expfile=NULL;
        int     cnt;


        if (expfile = fopen(ExpFileName,"r")){
                fscanf(expfile,"%d",&NumTreats);
                printf("Number of treatments: %d\n",NumTreats);
                for (cnt=0;cnt<NumTreats;cnt++)
                        expmod_readtreatment(expfile,&Treatments[cnt]);
                fclose(expfile);
        }
}

void    doexperimentmenu(){
        unsigned int treatment=0,cnt;
/*      noisy = 1;*/
#if     INDEPENDENT
        while (treatment<=NumTreats){
                printf("Choose a treatment\n");
                for (cnt=1;cnt<=NumTreats;cnt++)
                        printf("(%d) %s\n",cnt,Treatments[cnt-1].treatcode);
                printf("(%d) Exit\n",NumTreats+1);
                scanf("%d",&treatment);
                if (treatment<=NumTreats)
                        runexp(&Treatments[treatment-1],"janedoe");
                /*runexp(&Treatments[treatment-1],itsSInfo.subjName);*/
        }
#else
        /*code to choose appropriate experiment to run*/
        donexttreatment();
#endif
}



void load_configuration(char *config){
        FILE *cfgfile=NULL;

        /*cfgfile = fopen("vectexp.cfg","r");*/
        cfgfile = fopen(config,"r");
```

```
        if (cfgfile){
                fscanf(cfgfile,"%s",SceneDir);
                fscanf(cfgfile,"%s",ProfileDir);
                fscanf(cfgfile,"%s",SubjectDir);
                fscanf(cfgfile,"%s",ExpFileName);
                getcwd(cwd,100);
                cwd[strlen(cwd)+1]=0;
                cwd[strlen(cwd)]='\\';
        }
        else {
                printf("Error: Could not open configuration file.\nIs \"vectexp.cfg\" really
there?\n");
                exit(0);
        }
}

void donexttreatment(void){
        int next;
        char s[10];
        next = ((itsSInfo.treatment-1)+(itsSInfo.finished)) % NumTreats;
        printf("Now doing experiment %d of treatment %d (finished %d)\n",
             next+1,itsSInfo.treatment,itsSInfo.finished);
        printf("Subject's posture should be %s\n",
             (Treatments[next].posture==POSTURE_ERECT)?"ERECT":"SUPINE");

        /* put your trigger code here */
        /* comment out what is already here*/
        if (SERIAL_PORT_TRIGGER == 1) {
             itsSInfo.finished++;
             runexp(&Treatments[next],itsSInfo.subjName);
             subjdbUpdateFinished(&itsSInfo);
    }
        else {
                printf("Type 'G' to begin or 'X' to go back (and RETURN)\n");
                itsSInfo.finished++;
                scanf("%10s",s);

                if (s[0]!='X' && s[0]!='x'){
                        runexp(&Treatments[next],itsSInfo.subjName);
                        subjdbUpdateFinished(&itsSInfo);
                }
        }
}
```

subjdb.c

```
#include"wt.h"
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<direct.h>
#include<string.h>
#include"subjdb.h"
#include<sys\types.h>
#include<sys\stat.h>
#include"dirutils.h"
#include"expmod.h"
#include"statmod.h"
#include<time.h>

#define GET_CHAR newgetchar

enum {
        ksubjmenuMAIN,
        ksubjmenuSUBJECT,
        ksubjmenuNEWSUBJ,
        ksubjmenuDELETE,
        ksubjmenuCONTMENU,
        ksubjmenuSTATMENU,
        ksubjmenuRESETSUBJ,
        ksubjmenuCHOOSESUBJ
};

extern TreatmentInfoRec Treatments[];
static char moduleName[25]="Subject Database";

/*subjdb        globals                                 */

extern char             SubjectDir[80];
static int              subjectsLoaded;
static SubjectListRec   subjdbAllSubjects;
static ExpInfoRec       subjdbExpInfo;
static int              subjdb_initialized=0;
static char             subjdbExpFileName[1024];
static int              theChosen=0;
char                    SubjectDir[80];

/*Syntactic Sugar*/
#define subjects(j)     (subjdbAllSubjects.Subjects[(j)])
#define num_subjs       subjdbAllSubjects.numSubjs
#define subjname        ((theChosen) ? subjects(theChosen-1).subjName : "a subject.")

/*Decls*/
```

```
void DoQuit(void);
char newgetchar(void);
void subjdbInit(void);


/* a dirwalk function for finding all the cfg files */

#define MAX_CFGS 20
char cfg_files[MAX_CFGS][128];
int numcfgfiles=0;

void findcfg(char *name){
        if (strstr(name,".CFG"))
                strcpy(cfg_files[numcfgfiles++],name);
}

/*      For Choosing Subjects                   */



void main(int argc,char *argv[]){
        printf("clocks per sec %ld\n",CLOCKS_PER_SEC);
        subjdbMain();
}

void subjdbMain(void){
        /*Main entry point into this module             */
        /*perform any initialization if necessary       */
        if (!subjdb_initialized)
                subjdbInitialize();
        subjdbMainMenu();
        /*This is the last you should do when running a WTKapp*/
        printf("does this get run?\n");
        WTuniverse_delete();
}

void subjdbInitialize(void){
        /*Create the universe to be used*/
        WTuniverse_new(WTDISPLAY_STEREO, WTWINDOW_DEFAULT);
        subjectsLoaded=0;
        subjdbAllSubjects.numSubjs=0;
        subjdbExpInfo.numTreats=0;
        subjdbExpInfo.expName[0]=0;
        subjdbExpInfo.expDir[0]=0;
        subjdb_initialized=1;
}

/* Menus */
void subjdbMainMenu(){
```

```
        char choice=0;
        while (1) {
                subjdbPrintMenu(ksubjmenuMAIN);
                choice = GET_CHAR();
                switch (choice) {
                  case 'N':
                  case 'n':
                        subjdbNewExperiment();
                        break;
                  case 'L':
                  case 'l':
                        subjdbLoadExperiment();
                        break;
                  case 'S':
                  case 's':
                        subjdbSubjectMenu();
                        break;
                  case 'T':
                  case 't':
                        subjdbStatMenu();
                        break;
                  case 'Q':
                  case 'q':
                        DoQuit();
                        exit(0);
                  default:
                        break;
                }
        }
}

void subjdbSubjectMenu(){
        char choice=0;
        while ((choice!='X') && (choice!='x')){
                subjdbPrintMenu(ksubjmenuSUBJECT);
                choice = GET_CHAR();
                switch (choice){
                  case 'D':
                  case 'd':
                        subjdbDeleteSubject();
                        break;
                  case 'C':
                  case 'c':
                        subjdbContinueSubject();
                        break;
                  case 'B':
                  case 'b':
                        subjdbBackupSubject();
                        break;
```

```
                case 'N':
                case 'n':
                        subjdbNewSubject();
                        break;
                case 'R':
                case 'r':
                        subjdbResetSubject();
                        break;
                case 'P':
                case 'p':
                        subjdbChooseSubject();
                        break;
                default:
                  break;
              }
        }
}


void subjdbStatMenu(){
        char choice=0;
        while ((choice!='X') && (choice!='x')){
                subjdbPrintMenu(ksubjmenuSTATMENU);
                choice = GET_CHAR();
                switch(choice){
                  case 'C':
                  case 'c':
                        subjdbChooseSubject();
                        break;
                  case 'S':
                  case 's':
                        if (!theChosen){
                                subjdbChooseSubject();
                        }
                        if (theChosen){
                                subjdb_PerformStatsOnChosen();
                        }
                        break;
                  case 'A':
                  case 'a':
                        {       /*ALL*/
                                int savedChosen=theChosen;
                                subjdbLoadSubjectList();
                                for (theChosen=1;theChosen<=num_subjs;theChosen++)
                                        subjdb_PerformStatsOnChosen();
                                theChosen=savedChosen;
                        }
                        break;
                  default:
```

```
                    break;
                }
        }
}

void subjdbPrintMenu(int which){
        switch (which) {
          case ksubjmenuMAIN:
                printf("%s: Main Experiment Menu\n",moduleName);
                printf("Current Experiment is %s\n",subjdbExpInfo.expName);
                printf("(N)ew Experiment\n");
                printf("(L)oad other Experiment\n");
                printf("(S)ubject menu\n");
                printf("s(T)atistics menu\n");
                printf("(Q)uit\n");
                break;
          case ksubjmenuSUBJECT:
                printf("%s: Subject Menu\n",moduleName);
                if (theChosen) printf("Chosen subject %s\n",subjects(theChosen-
1).subjName);
                printf("(N)ew subject\n");
                printf("(D)elete %s\n", subjname);
                printf("(C)ontinue/Begin testing %s\n",subjname);
                printf("(B)ackup %s by one trial\n",subjname);
                printf("(R)eset %s\n",subjname);
                printf("(P)ick a subject.\n");
                printf("e(X)it, or back.\n");
                break;
          case ksubjmenuNEWSUBJ:
                printf("%s: New Subject Menu\n",moduleName);
                printf("(1-%d) Treatment type.\n",subjdbExpInfo.numTreats);
                printf("or 0 to exit back.");
                break;
          case ksubjmenuRESETSUBJ:
                printf("%s: Reset which subject?\n(0 to cancel)",moduleName);
                subjdbPrintAllSubjects();
                break;
          case ksubjmenuDELETE:
                printf("%s: Delete which subject?\n(0 to cancel)",moduleName);
                subjdbPrintAllSubjects();
                break;
          case ksubjmenuCONTMENU:
                printf("%s: Continue/Begin testing which subject?(0 to cancel)\n",
                    moduleName);
                subjdbPrintAllSubjects();
                break;
          case ksubjmenuCHOOSESUBJ:
                printf("%s: Choose a subject.\n",moduleName);
                subjdbPrintAllSubjects();
```

```
                break;
        case ksubjmenuSTATMENU:
                printf("%s: Stats Menu\n",moduleName);
                if (theChosen) {
                        printf("Chosen subject %s\n",subjects(theChosen-1).subjName);
                }
                printf("Perform (S)tats on %s.\n",subjname);
                printf("(C)hoose a subject.\n");
                printf("Perform Stats on (A)ll subjects.\n");
                printf("e(X)it\n");
                break;
        default:
                printf("%s: Unknown Menu", moduleName);
                break;
        }
}



void subjdbLoadExperiment(void){
        char    *cwd=NULL;
        int     whichexp;
        int     i;
        numcfgfiles = 0;
        cwd = getcwd(NULL,SUBJDB_MAXWDLEN);
        dirwalk(cwd,findcfg);
        if (numcfgfiles) {
                printf("Choose an experiment...\n");
again:          for (i=1;i<=numcfgfiles;i++)
                        printf("(%d) %s\n",i,cfg_files[i-1]);
                scanf("%d",&whichexp);
                if (whichexp<=numcfgfiles) {
                        strcpy(subjdbExpFileName,cfg_files[whichexp-1]);
                        expmod_setconfigfile(subjdbExpFileName);

                }
                else {
                        printf("You must choose a number between 1 and %d\n",
                            numcfgfiles);
                        goto again;
                }
        }
        else {
                printf("There are no experiments in this directory\n");
                printf("You must first create a new one.\n");
                return;
        }
        if (cwd) free(cwd);
}
```

```
void subjdbNewExperiment(void){
        /*not implemented yet*/
        printf("Not yet implemented\n");
}


void subjdbSaveExpInfo(void){
        /*not implemented yet*/
        printf("This isnt implemented yet.\n");

}


void subjdbDeleteSubject(void){
        int i,start,last;
        if (!theChosen) {
                printf("You have not chosen a subject to delete\n");
                return;
        }
        printf("Are you sure you want to delete %s(Y/N)?\n",subjname);
        start = theChosen;
        last = num_subjs--;
        for (i=start;i<last;i++)
                subjects(i-1) = subjects(i);
        subjdbWriteSubjects();
}

void subjdbChooseSubject(){
        /*entry point for choosing a subject*/
        theChosen = choosesubject(ksubjmenuCHOOSESUBJ);
}

void subjdbContinueSubject(){
        if (theChosen==0)
                theChosen = choosesubject(ksubjmenuCONTMENU);
        if (theChosen>0) {
                expmod_main(&subjects(theChosen-1));
        }

}
void subjdbBackupSubject(){
        if (theChosen != 0) {
                if (subjects(theChosen-1).finished > 0)
                        printf("Subject has completed %d runs.\n",--
subjects(theChosen-1).finished);
                else printf("Subject has not completed any runs.\n");
        }
}
```

```
void subjdbUpdateFinished(SubjectInfo *subjinfo){
        subjects(theChosen-1).finished = subjinfo->finished;
        subjdbWriteSubjects();
}

void subjdbResetSubject(){
        if (theChosen ==0)
                theChosen= choosesubject(ksubjmenuRESETSUBJ);
        subjects(theChosen-1).finished=0;
        subjdbUpdateFinished(&subjects(theChosen-1));
        printf("%s has been reset.\n",subjects(theChosen-1).subjName);

}
/* Subject Lists */

void subjdbLoadSubjectList(void){
        /*Loads a list of ALL test subjects (for a given
          experiment into an indexed datastructure
          for easy retrieval. Information loaded
          is only administrative-related info. It wont
          load in actual test results, only whether a
          user has completed a certain test and other
          relevant info. */

        /*make a quick exit if already loaded subjects, and
          delegate the work to another function*/

        if (!subjectsLoaded) subjectsLoaded = subjdbReadAllSubjs();

}

int     subjdbReadAllSubjs(void){
        printf("Reading all Subjs\n");
        char fname[512];
        FILE *subjfile=NULL;
        strcpy(fname,SubjectDir);
        strcat(fname,"subject.lst");

        printf("looking for subjects in %s\n",fname);
        if (subjfile = fopen(fname,"r")){
                int i;
                fscanf(subjfile,"%d",&subjdbAllSubjects.numSubjs);
                /*make sure num_subjs is less than maximum allowable*/
                num_subjs = (num_subjs<MAX_SUBJS) ? num_subjs : MAX_SUBJS;
                printf("Num subjects is %d\n",num_subjs);
                for (i=0;i<subjdbAllSubjects.numSubjs;i++)
                        subjdbReadSubjectInfo(subjfile,&subjdbAllSubjects.Subjects[i]);

                fclose(subjfile);
```

```
                return 1;
        }
        else {
                printf("Couldnt open %s\n",fname);
                return 0;
        }
}

void    subjdbReadSubjectInfo(FILE *subfile,SubjectInfo *aSub){
        /*read subject info into the subject record pointed to
          by *aSub. */
        fscanf(subfile,"%s",aSub->subjName);
        fscanf(subfile,"%d",&aSub->treatment);
        fscanf(subfile,"%d",&aSub->finished);
}

void subjdbWriteSubjects(void){
        char fname[512];
        FILE *subjfile=NULL;
        strcpy(fname,SubjectDir);
        strcat(fname,"subject.lst");


        if (subjfile = fopen(fname,"w")){
                int i;
                fprintf(subjfile,"%d\n",subjdbAllSubjects.numSubjs);
                for (i=0;i<subjdbAllSubjects.numSubjs;i++)
                        subjdbWriteSubjectInfo(subjfile,&subjdbAllSubjects.Subjects[i]);

                fclose(subjfile);
        }
}

void    subjdbWriteSubjectInfo(FILE *subfile,SubjectInfo *aSub){
        /*write subject info from the subject record pointed to
          by *aSub. */
        fprintf(subfile,"%s\n",aSub->subjName);
        fprintf(subfile,"%d\n",aSub->treatment);
        fprintf(subfile,"%d\n",aSub->finished);

}

void subjdbPrintAllSubjects(void){
        /*reads in all subjects into DB if necessary and then
          print out a list of them, with a number prepended so
          that the user can choose that subject. The number
          prepended corresponds to the index+1 in the loaded
          data structure. */
        int i;
```

```
        printf("Printing subjects, num_subjs = %d, subjectsLoades= %d\n",
            num_subjs,subjectsLoaded);
        if (!subjectsLoaded)
                subjdbLoadSubjectList();
        for (i=1;i<=num_subjs;i++)
                printf("(%2d) %8s(%1d,%1d)\n",i,subjects(i-1).subjName,
                    subjects(i-1).treatment,subjects(i-1).finished);

}

void subjdbNewSubject(void){
        char fname[512];
        SubjectInfo newsubj;
        if (num_subjs == MAX_SUBJS) {
                printf("Sorry, I dont have any room for more subjects.\n");
                printf("Complain to the author!\n");
                return;
        }
        subjdbLoadSubjectList();
        printf("Subject's name: ");
        scanf("%s",newsubj.subjName);
retry:  printf("%s's treatment(1-%d): ",newsubj.subjName,
            expmod_getnumtreats());
        scanf("%d",&newsubj.treatment);
        if (newsubj.treatment>expmod_getnumtreats() ||
           newsubj.treatment<1){
                printf("You must choose a value between 1-%d.\n",
                    expmod_getnumtreats());
                goto retry;
        }
        printf("New subject, %s, treatment %d, created.\n",
            newsubj.subjName, newsubj.treatment);
        newsubj.finished = 0;
        subjdbAllSubjects.Subjects[num_subjs++]=newsubj;
        /*NB: NEED TO CREATE A FOLDER IF NECESSARY!*/
        strcpy(fname,SubjectDir);
        strcat(fname,newsubj.subjName);
        mkdir(fname);
        subjdbWriteSubjects();
}

void subjdb_PerformStatsOnChosen(void){
        /*Create Text Data files corresponding to the
          subject specified by "theChosen" */

        int i;
        char temp[FNAMESIZE],expname[FNAMESIZE];
        printf("Doing Stats on %s...",subjname);
        for (i=1;i<=24;i++){
```

```
                strcpy(temp,SubjectDir);
                strcat(temp,subjname);
                temp[strlen(temp)+1]=0;
                temp[strlen(temp)]='\\';
                sprintf(expname,"%sf%d.dat",temp,i);
                statmod_init();
                if (!performstats(expname,&Treatments[i-1]))
                        printf("Could not open %s\n");

        }
        printf("...done.\n");
}

void DoQuit(void){
        /*If we need to do anything before quitting,
          do it here*/
        printf("Bbbddbdbdbdbbddbdb... That's all folks!\n");
        /*...nothing to do...*/
}

/*This picks the first character off a line*/

char newgetchar(void){
        char line[80];
        int i=0;
        scanf("%s",line);
        while (i<80 && (line[i]==' ' || line[i++]=='\t'));
        return (line[i-1]);
}

int  choosesubject(int menu){
        int choice;
        char line[80];
retry:  subjdbPrintMenu(menu);
        scanf("%s",line);
        choice = atoi(line);
        if (choice<0 || choice>num_subjs){
                printf("You must choose a number between 0-%d\n",
                     num_subjs);
                goto retry;
        }
        printf("You chose %d\n",choice);
        return choice;
}
```

Appendix B MATLAB Analysis Scripts

The following MATLAB scripts fall into two general categories: those used in the frequency analysis of the data, and those for the time-based analysis. A few programs were used for both. All of the code presented here was written by Michael Markmiller and Patricia Schmidt of the Man-Vehicle Lab, and those interested in using it in part or in its entirety must make reference to the writers, if only to gratify their prodigious egos.

Frequency Analysis:	Time Analysis
loadsub.m	CutLeadTail2.m
CutLeadTail2.m	tresp.m
csdtf.m	GSTimeHist4.m
bplot.m	thiststats.m
solvefortf.m	
varsf.m	
findtf2.m	

```
% LOADSUB  loads subject data file, files containing gseat pressure
%         and sled velocity profiles. Then calls findtf2 to get
%         csd ratios, averages and finds std of csd ratios for same sense and same
%         frequencies.  Calls solvefortf to solve for 3 transfer
%         functions.

%         This script uses findtf2.m, csdtf.m, solvefortf.m, bplot.m
%                   and cstd.m

clear;
clg


setfreq(1,:)=['sled=freq1;gseat=freq2;hmd=freq3;'];
setfreq(2,:)=['sled=freq2;gseat=freq1;hmd=freq3;'];
%setfreq(2,:)=['sled=freq3;gseat=freq1;hmd=freq2;'];
setfreq(3,:)=['sled=freq3;gseat=freq2;hmd=freq1;'];
%setfreq(3,:)=['sled=freq2;gseat=freq3;hmd=freq1;'];
setfreq(4,:)=['sled=freq2;gseat=freq3;hmd=freq1;'];
setfreq(5,:)=['sled=freq1;gseat=freq3;hmd=freq2;'];
setfreq(6,:)=['sled=freq3;gseat=freq1;hmd=freq2;'];
setfreq(7,:)=['sled=freq3;gseat=freq2;hmd=freq1;'];
setfreq(8,:)=['sled=freq2;gseat=freq1;hmd=freq3;'];
setfreq(9,:)=['sled=freq1;gseat=freq3;hmd=freq2;'];
setfreq(10,:)=['sled=freq3;gseat=freq1;hmd=freq2;'];
setfreq(11,:)=['sled=freq1;gseat=freq2;hmd=freq3;'];
setfreq(12,:)=['sled=freq2;gseat=freq3;hmd=freq1;'];


ldistfreq(1,:)=['load sled1;load hmd3'];
ldistfreq(2,:)=['load sled2;load hmd3'];
%ldistfreq(2,:)=['load sled3;load hmd2'];
ldistfreq(3,:)=['load sled3;load hmd1'];
%ldistfreq(3,:)=['load sled2;load hmd1'];
ldistfreq(4,:)=['load sled2;load hmd1'];
ldistfreq(5,:)=['load sled1;load hmd2'];
ldistfreq(6,:)=['load sled3;load hmd2'];
ldistfreq(7,:)=['load sled3;load hmd1'];
ldistfreq(8,:)=['load sled2;load hmd3'];
ldistfreq(9,:)=['load sled1;load hmd2'];
ldistfreq(10,:)=['load sled3;load hmd2'];
ldistfreq(11,:)=['load sled1;load hmd3'];
ldistfreq(12,:)=['load sled2;load hmd1'];

per=81.92;
freq2=1/per*[7 13 23 37];
freq3=1/per*[9 17 29 41];
freq1=1/per*[5 11 19 32];

subin(1,:)='ps';
subin(2,:)='dh';
subin(3,:)='ew';
subin(4,:)='jr';
subin(5,:)='kj';
subin(6,:)='mm';

%name='mmz';
%name = input('subject''s initials in lower case : ','s');
```

```
% loop through 6 subjects
for sub=1:3,
name=subin(sub,:);
name = [name,'z'];
for i=1:12
        i
        clear sled1 sled2 sled3 hmd1 hmd2 hmd3 p1 slider;
        fname=[name,num2str(i)];
        %cd ..
        cd G-Seat:GSData
        eval(['load G-Seat:GSData:',fname])
        %cd G-Seat:GSscripts
        %eval([fname,'=cutleadtail(',fname,');']);
        %cd G-Seat:GSData
        disp('loaded data file')
        eval(ldistfreq(i,:));
        disp('loaded profile file')
        eval(setfreq(i,:));
        data=eval(fname);
        slider=data(:,5);
        p1=data(:,1);
        clear eval(fname) data;
        [a1(i,:),a2(i,:),a3(i,:)]=findtf2(slider,vel,hmdv,p1,sled,gseat,hmd);
end % for i

% average a1, a2,a3 for same freqs by taking mean of real and imag. parts
a1mean1=mean([a1(1,:);a1(5,:);a1(9,:);a1(11,:)]);
a1mean2=mean([a1(2,:);a1(4,:);a1(8,:);a1(12,:)]);
a1mean3=mean([a1(3,:);a1(6,:);a1(7,:);a1(10,:)]);

%a1mean1=a1(1,:);
%a1mean2=a1(3,:);
%a1mean3=a1(2,:);

%a2mean1=a2(2,:);
%a2mean2=a2(1,:);
%a2mean3=a2(3,:);

%a3mean1=a3(3,:);
%a3mean2=a3(2,:);
%a3mean3=a3(1,:);

a2mean1=mean([a2(1,:);a2(5,:);a2(9,:);a2(11,:)]);
a2mean2=mean([a2(2,:);a2(4,:);a2(8,:);a2(12,:)]);
a2mean3=mean([a2(3,:);a2(6,:);a2(7,:);a2(10,:)]);

a3mean1=mean([a3(1,:);a3(5,:);a3(9,:);a3(11,:)]);
a3mean2=mean([a3(2,:);a3(4,:);a3(8,:);a3(12,:)]);
a3mean3=mean([a3(3,:);a3(6,:);a3(7,:);a3(10,:)]);

% standard deviation of complex number??
cd G-Seat:GSscripts
a1std1=cstd([a1(1,:);a1(5,:);a1(9,:);a1(11,:)]);
a1std2=cstd([a1(2,:);a1(4,:);a1(8,:);a1(12,:)]);
a1std3=cstd([a1(3,:);a1(6,:);a1(7,:);a1(10,:)]);

a2std1=cstd([a2(1,:);a2(5,:);a2(9,:);a2(11,:)]);
```

```
a2std2=cstd([a2(2,:);a2(4,:);a2(8,:);a2(12,:)]);
a2std3=cstd([a2(3,:);a2(6,:);a2(7,:);a2(10,:)]);

a3std1=cstd([a3(1,:);a3(5,:);a3(9,:);a3(11,:)]);
a3std2=cstd([a3(2,:);a3(4,:);a3(8,:);a3(12,:)]);
a3std3=cstd([a3(3,:);a3(6,:);a3(7,:);a3(10,:)]);

cd G-Seat:GSData
a3a=[a3mean1(1);a3mean2(1);a3mean3(1);a3mean1(2);a3mean2(2);a3mean3(2);a3mean1(3);a3mea
n2(3);a3mean3(3);a3mean1(4);a3mean2(4);a3mean3(4)];
a2a=[a2mean1(1);a2mean2(1);a2mean3(1);a2mean1(2);a2mean2(2);a2mean3(2);a2mean1(3);a2mea
n2(3);a2mean3(3);a2mean1(4);a2mean2(4);a2mean3(4)];
a1a=[a1mean1(1);a1mean2(1);a1mean3(1);a1mean1(2);a1mean2(2);a1mean3(2);a1mean1(3);a1mea
n2(3);a1mean3(3);a1mean1(4);a1mean2(4);a1mean3(4)];


a3s=[a3std1(1);a3std2(1);a3std3(1);a3std1(2);a3std2(2);a3std3(2);a3std1(3);a3std2(3);a3std3(3);a3std
1(4);a3std2(4);a3std3(4)];
a2s=[a2std1(1);a2std2(1);a2std3(1);a2std1(2);a2std2(2);a2std3(2);a2std1(3);a2std2(3);a2std3(3);a2std
1(4);a2std2(4);a2std3(4)];
a1s=[a1std1(1);a1std2(1);a1std3(1);a1std1(2);a1std2(2);a1std3(2);a1std1(3);a1std2(3);a1std3(3);a1std
1(4);a1std2(4);a1std3(4)];

a1b=a1a+a1s;
a2b=a2a+a2s;
a3b=a3a+a3s;

a1c=a1a-a1s;
a2c=a2a-a2s;
a3c=a3a-a3s;

freq=1/per*[5 7 9 11 13 17 19 23 29 32 37 41];
%cd ..
cd G-Seat:GSscripts
[E1cm, E1cp,E2cm,E2cp,E3cm,E3cp]= solvefortf(fname(1:2),a1a,a1s,a2a,a2s,a3a,a3s);
%[E1cmps, E1cpps,E2cmps,E2cpps,E3cmps,E3cpps]=solvefortf(a1b,a2b,a3b);
%[E1cmms, E1cpms,E2cmms,E2cpms,E3cmms,E3cpms]=solvefortf(a1c,a2c,a3c);

% save file containing a1a a2a a3a (means of a's at each freq),
% E1cm E1cp E2cm E2cp E3cm E3cp
% a1 a2 a3 (a's at each frequency)

%cd ..
%cd GSData
%filenum = input(['Enter number/letter to add to filename : ' name,'r'],'s');
filenum='p';
eval(['save G-Seat:GSResults:',fname(1:3),'r',filenum,' a1a a2a a3a E1cm E1cp E2cm E2cp E3cm
E3cp a1s a2s a3s a1 a2 a3']);

for clearloop =1:12
        eval(['clear ',fname(1:3),num2str(clearloop)]);
end % for clearloop

end %for sub
```

```
function [trimmed] = CutLeadTail2(untrimmed)
% CUTLEADTAIL2 Trims data leader and tail from original data. For GSeat experiment
% Uses section of leader on velocity data. Michael Markmiller 8/96

SAMPRATE = 50;                                          % Data sampling rate
TRIALLEN = 81.92;                                       % Trial length in sec
%VELOFFSET = -0.115;                                    % Mean noise value (if not zero mean)
VELOFFSET=mean(untrimmed(1:200,8));
VELSTD   = 0.0124;                                      % Std. Dev of noise
MINLEADER=200;
MAXLEADER = 3500;                                       % maximum leader before data
MAXTRAILER= 2000;
% find start. This looks for the first index outside the noise that is
% followed by at least 2 of 3 points outside noise on same side of noise band

abovenoise = find(untrimmed(1:MAXLEADER,8) > (VELOFFSET + 3*VELSTD));
for loop = 1:length(abovenoise)
        flag = 0;
        for innerloop = 1:3
                if (abovenoise(loop + innerloop) == abovenoise(loop) + innerloop);
                        flag = flag +1;
                end
        end
if (flag >=2)
        abovestart = abovenoise(loop);
        break;
end
end

belownoise = find(untrimmed(1:MAXLEADER,8) < (VELOFFSET - 3*VELSTD));
for loop = 1:length(belownoise)
        flag = 0;
        for innerloop = 1:3
                if (belownoise(loop + innerloop) == belownoise(loop) + innerloop);
                        flag = flag +1;
                end
        end
if (flag >=2)
        belowstart = belownoise(loop);
        break;
end
end

if (abovestart < belowstart)                    % Pick the earlier good signal
        istart = abovestart - 1;
else
        istart = belowstart - 1;
end

% find signal end. This will be checked against expected signal end
% discrepancies should be reported

abovenoise = find(untrimmed(length(untrimmed):-1:(length(untrimmed) - MAXTRAILER),8) >
(VELOFFSET + 3*VELSTD));
for loop = 1:length(abovenoise)
        flag = 0;
        for innerloop = 1:3
                if (abovenoise(loop + innerloop) == abovenoise(loop) + innerloop);
```

```
                        flag = flag +1;
                end
        end
if (flag >=2)
        aboveend = abovenoise(loop);
        break;
end
end

belownoise = find(untrimmed(length(untrimmed):-1:(length(untrimmed) - MAXTRAILER),8) <
(VELOFFSET - 3*VELSTD));

for loop = 1:length(belownoise)
        flag = 0;
        for innerloop = 1:3
                if (belownoise(loop + innerloop) == belownoise(loop) + innerloop);
                        flag = flag +1;
                end
        end
if (flag >=2)
        belowend = belownoise(loop);
        break;
end
end

if (aboveend < belowend)                % Pick the later good signal. Notice these indices count up
from end of untrimmed
        iend = length(untrimmed) - (aboveend - 1) + 1;
else
        iend = length(untrimmed) - (belowend - 1) + 1;
end

% plot velocity and show cutoff points
% click left of y-axis for yes, or
% click on desired starting point

hold off
plot(untrimmed(:,8),'y')
hold on
plot([istart,istart],[-1,1],'r')
plot([iend,iend],[-1,1],'r')
title('Cut off here?');
disp('Current start    end    length');
disp(['        ',num2str(istart),'    ',num2str(iend),'    ',num2str(iend-istart)]);
ylabel('click here for yes');
xlabel('or click on desired starting point on plot');
[x y]=ginput(1);
if x<0,
        end
end %if
if x>0,
        istart=x;
        iend=istart+TRIALLEN*SAMPRATE;
        plot([istart,istart],[-1,1],'g:')
        plot([iend,iend],[-1,1],'g:')
        title('Cut END off here?');
        [x y]=ginput(1);
        if x>0
```

```
                iend = x;
        end
end %if

% now cut ends off datafile. Don't forget to adjust time trace!

totaltime = (iend-istart + 1)/SAMPRATE;
disp(['Total time : ',num2str(totaltime)]);
if ((iend - istart)  > TRIALLEN*SAMPRATE)
        iend = istart + TRIALLEN*SAMPRATE;
        disp('Over expected length. Extra points truncated.');
end
trimmed = zeros(TRIALLEN*SAMPRATE + 1,10);
trimmed(1:((iend-istart)+1),1:9) = untrimmed(istart:iend,1:9);
trimmed(:,10) = (0:(1/SAMPRATE):(TRIALLEN))';


end
```

```
function [mag, phase, freq]=csdtf(in,out,d,nfft,fs,window)
%  CSDTF Finds transfer function between x and y by cross-correlation

[Pxd, Fxd]=csd(in,d,nfft,fs,window);
[Pyd,Fyd]=csd(out,d,nfft,fs,window);

Pxdmag=abs(Pxd);
Pxdphase=atan2(imag(Pxd),real(Pxd));

Pydmag=abs(Pyd);
Pydphase=atan2(imag(Pyd),real(Pyd));

mag=Pydmag ./Pxdmag;
phase=Pydphase-Pxdphase;
freq=Fxd;



function bplot(mag,phase,w,lc)
% BPLOT.m makes bode plot from mag, phase(deg), freq(rad/sec) data

subplot(211)
%cla
loglog(w,mag,lc)
hold on
loglog(w,mag,lc)
%axis([.01,1.,.01,100])
grid on
ylabel('log gain')
xlabel('log freq (Hz)')

subplot(212)
%cla
semilogx(w,phase,lc)
hold on
%axis([.01,1,-5,5])
grid on
semilogx(w,phase,lc)
ylabel('phase (deg)')
xlabel('log freq (Hz)')
```

```
function
[E1cm,sigmamE1,E1cp,sigmapE1,E2cm,sigmamE2,E2cp,sigmapE2,E3cm,sigmamE3,E3cp,sigmapE3]
=solvefortf(name,a1a,a1s,a2a,a2s,a3a,a3s)
%% SOLVEFORTF.m  uses block diagram eqns to solve for tf's

per=81.92;
freq=1/per*[5 7 9 11 13 17 19 23 29 32 37 41]';
% solve for estimator tf's

%eqns for old block diagram
%E1c=a1+a2./(1-a2.*K.*P);
%E2c=-1*a2./(1-a2.*K.*P);
%E3c=a3+P.*K.*a1.*a3;

% joystick gain and low-pass filter
K=.08./((j*freq*2*pi)+10);

% sled dynamics are flat for this frequency range
P=1;

% eqns for 8/19 block diagram, not including otolith and
% body-seat compression dynamics
%E1c=a1a./(j.*freq*2*pi*P);
%E2c=a2a.*(1+a1a.*K);
%E3c=a3a.*(1+a1a.*K);

% revision 9/2/96
E1c=-a1a./(j.*freq*2*pi*P);
E2c=a2a.*(-1+a1a.*K);
E3c=a3a.*(1-a1a.*K);

% find real and imaginary parts of variances of a's
vra1=(real(a1s).^2);
via1=(imag(a1s).^2);

vra2=(real(a2s).^2);
via2=(imag(a2s).^2);

vra3=(real(a3s).^2);
via3=(imag(a3s).^2);


%standard deviations of mag and phase of E's given std's of a's
sigmarE1=sqrt((1)*via1./(2*pi*freq));
sigmaiE1=sqrt((1)*vra1./(2*pi*freq));

sigmarE2=sqrt(abs(real(((E2c)./(a2a)).^2).*vra2+real((K).*(E2c).^2./(1+(K).*(a1a)).^2).*vra1));
sigmaiE2=sqrt(abs(imag(((E2c)./(a2a)).^2).*via2+imag((K).*(E2c).^2./(1+(K).*(a1a)).^2).*via1));

sigmarE3=sqrt(abs(real(((E3c)./(a3a)).^2).*vra3+real((K).*(E3c).^2./(1+(K).*(a1a)).^2).*vra1));
sigmaiE3=sqrt(abs(imag(((E3c)./(a3a)).^2).*via3+imag((K).*(E3c).^2./(1+(K).*(a1a)).^2).*via1));

sigmamE1=abs(sigmarE1+j*sigmaiE1);
sigmapE1=atan2(sigmaiE1,sigmarE1);

sigmamE2=abs(sigmarE2+j*sigmaiE2);
sigmapE2=atan2(sigmaiE2,sigmarE2);
```

```
sigmamE3=abs(sigmarE3+j*sigmaiE3);
sigmapE3=atan2(sigmaiE3,sigmarE3);

% convert to mag, phase
E1cm=abs(E1c');
%E1cp=rem(atan2(imag(E1c),real(E1c)),360)
E1cp=rem(phase(E1c')*180/pi,360);
E2cm=abs(E2c');
%E2cp=rem(atan2(imag(E2c),real(E2c)),360)
E2cp=rem(phase(E2c')*180/pi,360);
E3cm=abs(E3c');
%E3cp=rem(atan2(imag(E3c),real(E3c)),360)
E3cp=rem(phase(E3c')*180/pi,360);


% plot tf's
% o=vestibular  x= tactile  + = visual

orient tall
clg
bplot(E1cm,E1cp,freq,'o')
bplot(E1cm,E1cp,freq,'-')
bplot(E1cm+sigmamE1',E1cp+sigmapE1'*180/pi,freq,'+')
bplot(E1cm-sigmamE1',E1cp-sigmapE1'*180/pi,freq,'+')

subplot(211)
eval(['title("subject ',name(1:2),'                    vestibular response',"")'])


hold on
for i=1:12
        subplot(212)
        semilogx([freq(i) freq(i)],([E1cp(i)+sigmapE1(i)*180/pi E1cp(i)-sigmapE1(i)*180/pi]),'-')
        subplot(211)
        if E1cm(i)<sigmamE1(i),
                loglog([freq(i) freq(i)],([E1cm(i) E1cm(i)+sigmamE1(i)]),'-');
        else
        loglog([freq(i) freq(i)],([E1cm(i)+sigmamE1(i) E1cm(i) - sigmamE1(i)]),'-')
        end %if
end %for

%pause;
%print -dps jrvest.eps


clg
bplot(E2cm,E2cp,freq,'x')
bplot(E2cm+sigmamE2',E2cp+sigmapE2'*180/pi,freq,'+')
bplot(E2cm-sigmamE2',E2cp-sigmapE2'*180/pi,freq,'+')
bplot(E2cm,E2cp,freq,'-')
subplot(211)
eval(['title("subject ',name(1:2),'                    tactile response',"")'])

for i=1:12
        subplot(212)
        semilogx([freq(i) freq(i)],([E2cp(i)+sigmapE2(i)*180/pi E2cp(i)-sigmapE2(i)*180/pi]),'-')
        subplot(211)
```

```
        if E2cm(i)<sigmamE2(i),
                loglog([freq(i) freq(i)],([E2cm(i) E2cm(i)+sigmamE2(i)]),'-');
        else
                loglog([freq(i) freq(i)],([E2cm(i)+sigmamE2(i) E2cm(i) - sigmamE2(i)]),'-')
        end %if
end %for

%pause;
%print -dps ewtact.eps
clg

bplot(E3cm,E3cp,freq,'o')
bplot(E3cm,E3cp,freq,'-')
bplot(E3cm+sigmamE3',E3cp+sigmapE3'*180/pi,freq,'+')
bplot(E3cm-sigmamE3',E3cp-sigmapE3'*180/pi,freq,'+')
subplot(211)
eval(['title("subject ',name(1:2),'                              visual response','")'])

for i=1:12
        subplot(212)
        semilogx([freq(i) freq(i)],([E3cp(i)+sigmapE3(i)*180/pi E3cp(i)-sigmapE3(i)*180/pi]),'-')
        subplot(211)
        if E3cm(i)<sigmamE3(i),
                loglog([freq(i) freq(i)],[E3cm(i) E3cm(i)+sigmamE3(i)],'-');
        else
                loglog([freq(i) freq(i)],([E3cm(i)+sigmamE3(i) E3cm(i) - sigmamE3(i)]),'-')
                disp('whole errorbar')
        end %if
end %for
%print -dps dhvis.eps
```

```
function [p1,p2,p3,p4,slider,comm,acc,vel,pos,time] = varsf(data)
%%% VARSF Function version of vars.m variables for findtf.m

p1=data(:,1);
p2=data(:,2);
p3=data(:,3);
p4=data(:,4);
slider=data(:,5);
comm=data(:,6);
acc=data(:,7);
vel=data(:,8);
pos=data(:,9);
time=data(:,10);



function [a1, a2, a3]= findtf2(slider,vel,hmdv,p1,sled,gseat,hmd)
%%% FINDTF2 uses csd's to find csd ratios at stimulus freqs
%%% calls csdtf.m and bplot.m

% set tolerance for frequency and input frequencies
tol=.015;
per=81.92;
%freq=1/per*[5 7 9 11 13 17 19 23 29 32 37 41];
freq2=1/per*[7 13 23 37];
freq3=1/per*[9 17 29 41];
freq1=1/per*[5 11 19 32];
K=.08;

vel=vel';
hmdv=hmdv';

%freqs=freq3;
%freqg=freq2;

% pad with zeros to make data,profiles same length
if length(vel)<length(slider),
        disp('adding zeros to vel, hmdv')
        vel=[vel;zeros(length(slider)-length(vel),1)];
        hmdv=[hmdv;zeros(length(slider)-length(hmdv),1)];
        length(vel)
        length(hmdv)
end % if length

if length(slider)<length(vel),
        slider=[slider;zeros(length(vel)-length(slider),1)];
        disp('adding zeros to p1, slider')
        p1=[p1;zeros(length(vel)-length(p1),1)];
end % if length

% find ratios of csd's
[a1m, a1p,f1]=csdtf(vel,slider,vel,length(vel),50,[]);
%bplot(a1m,a1p,f1,'o')
[a2m,a2p,f2]=csdtf(p1,slider,p1,length(p1),50,[]);
%bplot(a2m,a2p,f2,'x')
[a3m,a3p,f3]=csdtf(hmdv,slider,hmdv,length(hmdv),50,[]);
%bplot(a3m,a3p,f3,'+')
```

```
disp('took csds')

% throw away csd data for freq>1 hz
f1=f1(1:find(f1>1));
f2=f2(1:find(f2>1));
f3=f3(1:find(f3>1));

a1m=a1m(1:find(f1>1));
a2m=a2m(1:find(f2>1));
a3m=a3m(1:find(f3>1));

a1p=a1p(1:find(f1>1));
a2p=a2p(1:find(f2>1));
a3p=a3p(1:find(f3>1));

% find mags and phases at input frequencies by averaging over
% frequencies within tolerance
a1a=zeros(1,8);
a2a=a1a;
a3a=a1a;

% convert back to a+bj form
a1a=a1m.*(cos(a1p)+j*sin(a1p));
a2a=a2m.*(cos(a2p)+j*sin(a2p));
a3a=a3m.*(cos(a3p)+j*sin(a3p));
%a3a(i)=a3ma(i)*(cos(a3pa(i))+j*sin(a3pa(i)));

for i=(1:4),

%  *** average  a+bj
        a1(i)=mean(a1a(find(abs(f1-sled(i))<tol)));
        %a1p(i)=mean(a1p(find(abs(f1-freq(i))<tol)));
        %f1(find(abs(f1-freqs(i))<tol))

        a2(i)=mean(a2a(find(abs(f2-gseat(i))<tol)));
        %a2pa(i)=mean(a2p(find(abs(f2-freq(i))<tol)));
        %f1(find(abs(f1-freqg(i))<tol))

        a3(i)=mean(a3a(find(abs(f3-hmd(i))<tol)));
        %a3ma(i)=mean(a3m(find(abs(f3-freq(i))<tol)));
        %a3pa(i)=mean(a3p(find(abs(f3-freq(i))<tol)));
end % for i

 disp('after for loop')
 P=1;  %******fix this*****
% solve for estimator tf's ***change to least-squares fit ***
%E1c=a1a+a2a./(1-a2a.*K.*P);
%E2c=-1*a2a./(1-a2a.*K.*P);
%E3c=a3+P.*K.*a1.*a3;

% convert to mag, phase
%E1cm=abs(E1c)
%E1cp=atan2(imag(E1c),real(E1c))
%E2cm=abs(E2c)
%E2cp=atan2(imag(E2c),real(E2c))
%E3cm=abs(E3c)
%E3cp=atan2(imag(E3c),real(E3c))
```

```
%%% TRESP.m  cuts off leader and tail from gseat data file
%%% calls cutleadtail2.m and plots for confirmation and correction
%%% then call gstimehist4 to get time response data. Results are
%%% saved
clear;

%%% hmd profiles
hmdprof(1,:)=['hmd3'];
hmdprof(2,:)=['hmd3'];
hmdprof(3,:)=['hmd1'];
hmdprof(4,:)=['hmd1'];
hmdprof(5,:)=['hmd2'];
hmdprof(6,:)=['hmd2'];
hmdprof(7,:)=['hmd1'];
hmdprof(8,:)=['hmd3'];
hmdprof(9,:)=['hmd2'];
hmdprof(10,:)=['hmd2'];
hmdprof(11,:)=['hmd3'];
hmdprof(12,:)=['hmd1'];

timeres=zeros(12,18);
threeways=zeros(8,3);

subin(1,:)='ps';
subin(2,:)='mm';
subin(3,:)='ew';
subin(4,:)='jr';
subin(5,:)='kj';
subin(6,:)='dh';

%%% filter coefficients for pressure data
a=[1 -2.4986 2.1153 -.6041];
b=[.0016 .0047 .0047 .0016];

SaveAsNum = input('input number to add to saved results files: xxtr','s');
% loop through 6 subjects
for sub=1:6,
name=subin(sub,:);
%name = input('subject''s initials in lower case : ','s');
name = [name,'z'];

i = 1;
while i<13
%for i=1:12
        if sub == 6
                while ((i==4) | (i==6) | (i==7) | (i==8) | (i==9))
                        i = i+1;
                end % while to skip trials
                if (i==12)
                        break
                end
        end
        fname=[name,num2str(i)];
        cd G-Seat:GSData
        eval(['load G-Seat:GSData:',fname])
        disp(['loaded ',fname])
```

```
        %%% truncate file if necessary
        eval(['[row col]=size(',fname,');']);
        if row > 4100,
                cd G-Seat:GSscripts
                eval(['VELOFFSET = mean(',fname,'(1:100,8));']);
                eval(['ACCOFFSET = mean(',fname,'(1:100,7));']);
                eval([fname,'=cutleadtail2(',fname,');']);
                cd G-Seat:GSData
                trflag=1;
        else
                disp([fname,' already truncated'])
                eval(['VELOFFSET = mean(',fname,'(:,8));']);
                eval(['ACCOFFSET = mean(',fname,'(:,7));']);
        end %if row


        eval(['load G-Seat:GSData:',hmdprof(i,:)])
        eval(['slider = ',fname,'(:,5);']);
        eval(['sa = ',fname,'(:,7);']);
        eval(['sv = ',fname,'(:,8);']);
        eval(['p1 = ',fname,'(:,1);']);
        eval(['t = ',fname,'(:,10);']);
        hv=hmdv';
        hv(length(slider)+1:length(hv))=[];

        deltat=mean(diff(t));

        ga=filtfilt(b,a,p1);
        gj=[diff(ga)/deltat;0];

        ha=[diff(hv)/deltat;0];

        [tr,threew]=gstimehist4(slider,sv,sa,ga,gj,hv,ha,VELOFFSET,ACCOFFSET);
        timeres=timeres+tr;
        threeways=threeways+threew;

        if trflag==1,
                eval(['save ',fname,' ',fname,' -ascii'])
                trflag=0;
        end %if trflag
        eval([fname,' = 0;']);
        i = i +1;
end % while i
%filenum = input(['enter number to add to filename ',fname(1:3),'tr']);
%savefile = [fname(1:3),'tr',num2str(filenum)];
savefile = [fname(1:3),'tr',SaveAsNum];

        eval(['save G-Seat:GSData:',savefile,' timeres threeways'])
for clearloop =1:12
        eval(['clear ',fname(1:3),num2str(clearloop)]);
end % for clearloop
end % for sub
```

```
function [timeresponses,threeways] = gstimehist4(slider,sv,sa,ga,gj,hv,ha,VELOFFSET,ACCOFFSET)
% GSTIMEHIST4 Analyzes the time-domain slider response of a given trial. It finds
% the number of data points that the subject acted to null a given
% stimulus (or combination of stimuli), did not act, or issued a command
% contrary to the given stimulus. Michael Markmiller 8/27/96

%sa = -sa;                      % ??Accelerometer is reverse from what we thought! MM 9/3/96
%ACCOFFSET = -ACCOFFSET;

SVTHRESHPOS = VELOFFSET + 0.03;        % Sled velocity and accel thresholds.
SVTHRESHNEG = VELOFFSET - 0.03;        % Slider responses between the thresholds
SATHRESHPOS = ACCOFFSET + 0.023;       %4.89 are considered to be "non-responses".
SATHRESHNEG = ACCOFFSET - 0.023;       %4.84 These are in voltages, not SI units

GATHRESHPOS = -2.9;                    % Same as above for GSeat accel and jerk.
GATHRESHNEG = -3.1;
GJTHRESHPOS =  0.1;
GJTHRESHNEG = -0.1;

HVTHRESHPOS =  0.015;                  % Same as above for HMD velocity and accel.
HVTHRESHNEG = -0.015;                  % HMD velocities and accel seem to be in m/s or
m/s2
HATHRESHPOS =  0.015;                  % Based on 1 pixel/sec
HATHRESHNEG = -0.015;

POSRESP             = 0.3;
NEGRESP             = -0.3;

THRESHOLDS(1,:) = [SVTHRESHPOS SATHRESHPOS GATHRESHPOS GJTHRESHPOS
HVTHRESHPOS HATHRESHPOS];
THRESHOLDS(2,:) = [SVTHRESHNEG SATHRESHNEG GATHRESHNEG GJTHRESHNEG
HVTHRESHNEG HATHRESHNEG];

TIMEDELAY = 0;                         % Number of data points considered to be
                                       % the subject's reaction time. NOT USED. SEE DELAYS
TRIALLENGTH = 4097;                    % Nominal trial length (# of data points)
                                % --fileno--
%delays=[0,0.6,0.12,0.12,0.2,0.2];  % #1   time delay in sec for each column in profile
%delays=[0.2,0.2,0.2,0.2,0.2,0.2];  % #2, #5
delays=[0,0,0,0,0,0];               % #3,#4(after accel inversion), #6(noninverted)
%delays=[0.1,0.1,0.1,0.1,0.1,0.1];  %
deltat=1/50;                        % sample rate

profiles = [sv sa ga gj hv ha];

%%%%%%%% Determine if data file has been truncated

if (length(slider) > (TRIALLENGTH + 5))
        disp('WARNING : Trial has not been truncated. Skipping.');
        break;
end

%%%%%%%%  shift data to remove time delays
for i=1:6,
        profiles(:,i) = [zeros(round(delays(i)/deltat),1) ; profiles(1:(length(profiles(:,i)) -
round(delays(i)/deltat)),i)];
end % for i
```

```
%%%%%%%% find the appropriate indices of positive/ negative signals

svp = find(sv > SVTHRESHPOS);       % The data matrix "timeresponses"
svn = find(sv < SVTHRESHNEG);       % has these rows and columns:
sap = find(sa > SATHRESHPOS);
san = find(sa < SATHRESHNEG);       %       SV    SA    GA    GJ    HV    HA
                                    % SV  [                                  ]
gap = find(ga > GATHRESHPOS);       % SA  [                                  ]
gan = find(ga < GATHRESHNEG);       % GA  [                                  ]
gjp = find(gj > GJTHRESHPOS);       % GJ  [                                  ]
gjn = find(gj < GJTHRESHNEG);       % HV  [                                  ]
                                    % HA  [                                  ]
hvp = find(hv > HVTHRESHPOS);
hvn = find(hv < HVTHRESHNEG);
hap = find(ha > HATHRESHPOS);
han = find(ha < HATHRESHNEG);

%%% Debugging section

figure(1);
clf;
subplot(2,1,1);
plot(sv,':');
hold on;
plot(svp,sv(svp),'+');
plot(svn,sv(svn),'o');
hold off;
subplot(2,1,2);
plot(sa,':');
hold on;
plot(sap,sa(sap),'+');
plot(san,sa(san),'o');
hold off;
disp(['VELOFFSET = ' num2str(VELOFFSET)]);
disp(['mean vel  = ' num2str(mean(sv))]);
disp(['ACCOFFSET = ' num2str(ACCOFFSET)]);

%%% End debugging section

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% The following loops construct a matrix of responses to
%%%%%%% single modes (eg positive sled accel) and two-mode
%%%%%%% interactions. Each 2 x 3 block is an accounting of the
%%%%%%% response data points that occured while the given condition
%%%%%%% was true. Single conditions (along the diagonal) and confirming
%%%%%%% cues (upper half) can be read as follows (each 2 x 3 block):
%%%%%%% "positive cue"    [(NULLING RESPONSE) (NO RESP) (INCORRECT RESPONSE)]
%%%%%%% "negative cue"    [(NULLING RESPONSE) (NO RESP) (INCORRECT RESPONSE)]
%%%%%%%
%%%%%%% Conflicting cue interactions (lower half of matrix) are read
%%%%%%% in relation to the cue listed for that row in the matrix:
%%%%%%% "pos OR neg"      [(NULLING RESPONSE) (NO RESP) (INCORRECT RESPONSE)]
%%%%%%%                   [         0             0              0          ]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


timeresponses = zeros(12,18);
```

```
%%% Upper half : single and confirming cues

for rowloop = 1:6

        for columnloop = rowloop:6
                        posnullbin = 0;
                        posnonebin = 0;
                        posincobin = 0;
                        negnullbin = 0;
                        negnonebin = 0;
                        negincobin = 0;
                        %for counter = 1:(length(slider) - TIMEDELAY)

                                posnullbin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop)))));

                                posincobin=length(find(((slider > POSRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop)))));

                                posnonebin=length(find(((slider > NEGRESP) & ...
                                        (slider < POSRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop)))));

                                negnullbin=length(find(((slider > POSRESP) & ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop)))));

                                negincobin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop)))));

                                negnonebin=length(find(((slider > NEGRESP) & ...
                                        (slider < POSRESP) & ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop)))));

                                %end    % if
                        %end    % for counter loop

                        timeresponses((2*rowloop - 1),(3*columnloop -2):(3*columnloop)) = ...
                                                [posnullbin posnonebin posincobin];
                        timeresponses((2*rowloop),(3*columnloop -2):(3*columnloop)) = ...
                                                [negnullbin negnonebin negincobin];
        end % for columnloop
end % for rowloop

%%% End of Upper half
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Lower half : conflicting cues : see description

for rowloop = 2:6

        for columnloop = 1:(rowloop - 1)
                rownullbin      = 0;
                nonebin         = 0;
                colnullbin      = 0;

                rownullbin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop))))) + ...
                                        length(find(((slider > POSRESP) & ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop)))));

                colnullbin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop))))) + ...
                                        length(find(((slider > POSRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop)))));

                nonebin   =length(find(((slider > NEGRESP) & (slider < POSRESP) & ...
                                        (profiles(:,rowloop) > THRESHOLDS(1,rowloop)) & ...
                                        (profiles(:,columnloop) <
THRESHOLDS(2,columnloop))))) + ...
                                        length(find(((slider > NEGRESP) & (slider < POSRESP)
& ...
                                        (profiles(:,rowloop) < THRESHOLDS(2,rowloop)) & ...
                                        (profiles(:,columnloop) >
THRESHOLDS(1,columnloop)))));

                timeresponses((2*rowloop - 1),(3*columnloop -2):(3*columnloop)) = ...
                                                [rownullbin nonebin colnullbin];
                timeresponses((2*rowloop),(3*columnloop -2):(3*columnloop)) = ...
                                                [0 0 0];
        end % for columnloop
end % for rowloop

%%% End of Lower Half

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%         IMPLEMENTED BUT NOT USED
%%%%%%%         Three-way confirming cues. Here we are only interested in
%%%%%%%         the following combinations:
%%%%%%%
%%%%%%%         1. SV GA HV
%%%%%%%         2. SA GA HA
%%%%%%%         3. SA GJ HA
%%%%%%%         4. SA GA HV
%%%%%%%
```

```
%%%%%%%          The format is the same as the upper half of previous matrix
%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

threeways = zeros(8,3);                          % Where the data will go
prfpoint = [1 3 5 ; 2 3 6 ; 2 4 6 ; 2 3 5];    % Letting the loop parse arbitrary profiles

for loop = 1:4;
        posnullbin = 0;
        posnonebin = 0;
        posincobin = 0;
        negnullbin = 0;
        negnonebin = 0;
        negincobin = 0;

        posnullbin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) >
THRESHOLDS(1,prfpoint(loop,1))) & ...
                                        (profiles(:,prfpoint(loop,2)) >
THRESHOLDS(1,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) >
THRESHOLDS(1,prfpoint(loop,3))))));

        posincobin=length(find(((slider > POSRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) >
THRESHOLDS(1,prfpoint(loop,1))) & ...
                                        (profiles(:,prfpoint(loop,2)) >
THRESHOLDS(1,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) >
THRESHOLDS(1,prfpoint(loop,3))))));

        posnonebin=length(find(((slider < POSRESP) & (slider > NEGRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) >
THRESHOLDS(1,prfpoint(loop,1))) & ...
                                        (profiles(:,prfpoint(loop,2)) >
THRESHOLDS(1,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) >
THRESHOLDS(1,prfpoint(loop,3))))));

        negnullbin=length(find(((slider > POSRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) <
THRESHOLDS(2,prfpoint(loop,1))) & ...
                                        (profiles(:,prfpoint(loop,2)) <
THRESHOLDS(2,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) <
THRESHOLDS(2,prfpoint(loop,3))))));

        negincobin=length(find(((slider < NEGRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) <
THRESHOLDS(2,prfpoint(loop,1))) & ...
                                        (profiles(:,prfpoint(loop,2)) <
THRESHOLDS(2,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) <
THRESHOLDS(2,prfpoint(loop,3))))));

        negnonebin=length(find(((slider < POSRESP) & (slider > NEGRESP) & ...
                                        (profiles(:,prfpoint(loop,1)) <
THRESHOLDS(2,prfpoint(loop,1))) & ...
```

```
                                        (profiles(:,prfpoint(loop,2)) <
THRESHOLDS(2,prfpoint(loop,2))) & ...
                                        (profiles(:,prfpoint(loop,3)) <
THRESHOLDS(2,prfpoint(loop,3))))));

        threeways((2*loop - 1),:) = [posnullbin posnonebin posincobin];
        threeways((2*loop),:)     = [negnullbin negnonebin negincobin];

end % for loop

end % function
```

```
%%%%% THISTSTATS.m loads files saved by gstimehistx.m, combines positive
%%%%% and negative rows of timeresponses matrix, finds means and stdev's
%%%%% of each element over all subjects, and computes t and d, the degrees
%%%%% of freedom.


subin(1,:)='ps';
subin(2,:)='mm';
subin(3,:)='ew';
subin(4,:)='jr';
subin(5,:)='kj';
subin(6,:)='dh';

trmean=zeros(6,18);
trstd=trmean;
trvar=trmean;
t=zeros(6,6);

for sub=1:6,
        name=subin(sub,:);
        fname=[name,'ztr6'];
        eval(['load G-Seat:GSData:',fname])
        disp(['loaded ',fname])
        %eval([name,'tr = timeres;'])

        for i=(1:4)
                eval(['tr',name,'(i,:)=timeres(2*i-1,:)+timeres(2*i,:);'])
        end %for i

        eval(['trmean=1/6*trmean+tr',name,';']);
end % for sub


for sub=1:6,
        name=subin(sub,:);
        eval(['trvar=trstd+1/5*(tr',name,' - trmean).^2;'])
end %for sub

trstd=sqrt(trvar);

for col=1:1
        t(:,col)=(trmean(:,3*col-2)-trmean(:,3*col))./(sqrt(trvar(:,3*col-2)+trvar(:,3*col))/sqrt(6));
        %dprime(:,col)=5*((trvar(:,3*col-2)+trvar(:,3*col))/6).^2./(((trvar(:,3*col-
2)).^2+(trvar(:,3*col)).^2)/6);
        dprime(:,col)=5*(trvar(:,3*col-2)+trvar(:,3*col)).^2./((trvar(:,3*col-
2)).^2+(trvar(:,3*col)).^2);
        d(:,col)=floor(dprime(:,col));
end % for col
```

Appendix C Visual Model and Motion Profiles

The three dimensional visual model used in this research was created using 3DStudio, from AutoDesk Corp. The model was based on measurements taken from the room in which the experiments were performed. Once the room model had been completed, it was imported as a .3DS file into the WTK Demo, which is a stand alone scene viewer provided with WTK. From there it was saved in Neutral File Format (.NFF), which is an ASCII representation of the room. This format lists all the room vertices and their XYZ coordinates first, then specifies the polygons defined by those vertices. WTK supports quadrilateral (4-sided) polygons, and the author found that rendering was substantially faster when these were used in place of the standard triangles. Therefore, all the room surfaces were changed by hand from pairs of triangles to rectangles.

Following the scene description are the experiment text files that were loaded by the visual scene program described in Appendix A. The Load Experiment menu reads the sledz.cfg file and displays it as an option. The sledz2.txt file contains motion and scene specifics for each trial in the experiment. Each motion profile must follow this format:

1	The number of "movements."
100	Type of movement. 100 specifies linear motion along room's long axis.
2048	# of data points in movement.
0.0	First data point. Each point is a viewpoint position, and represents one
1.25	frame of animation.
2.50	
3.0	
.	
.	
.	
2.17	Last point in movement.
14 xx xx	Next movement type, if desired. Currently, only one linear movement per trial is supported. Angular movements may be used as well.

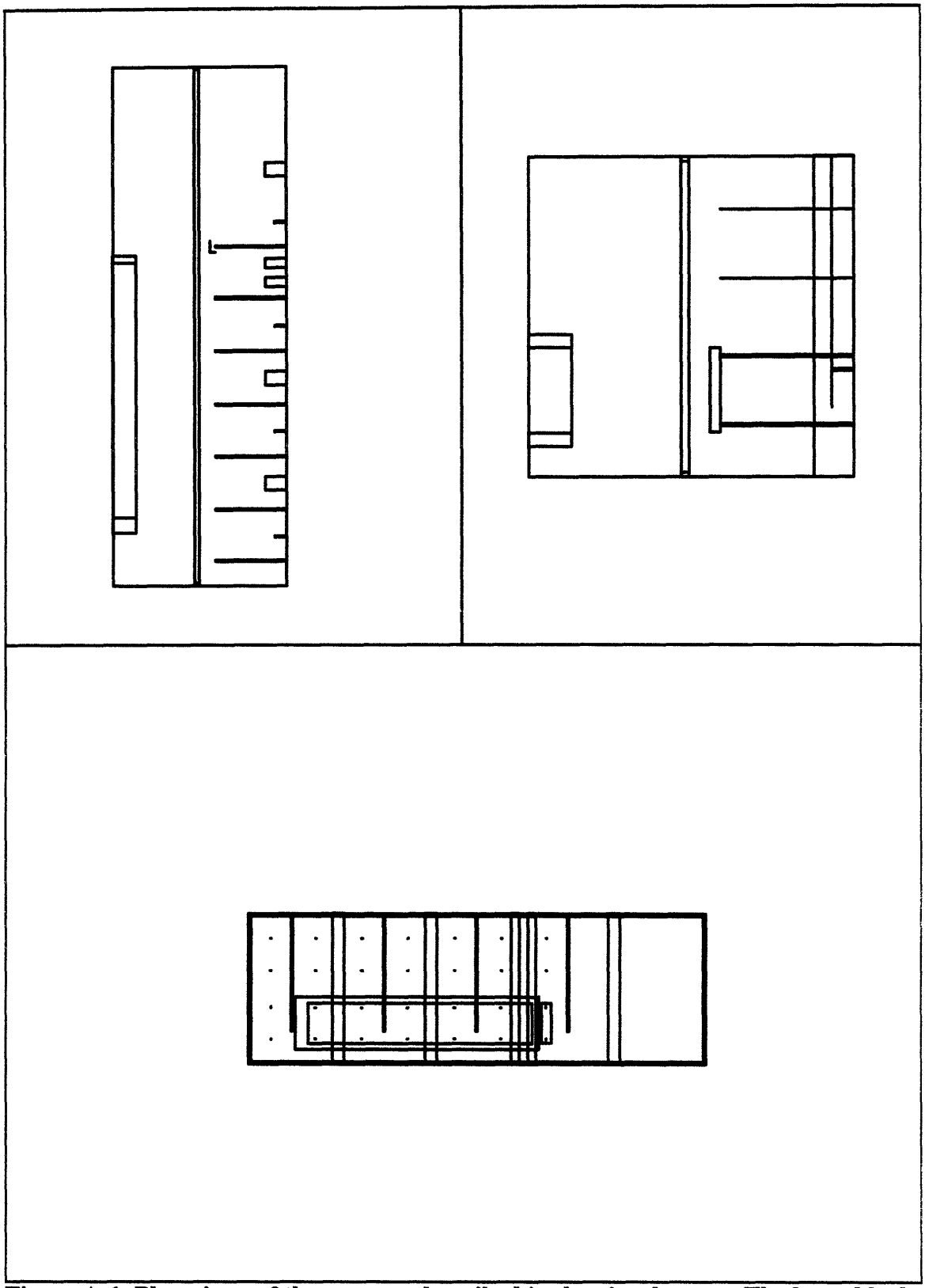


Figure A-1 Plan views of the room as described in the visual scene. The large block on the floor of the room is the Sled base, upon which the "virtual cart" translates.

```
nff // sldrmz5.nff
version 2
//viewpos -120.075569 357.837646 -6.68862
viewpos 820.0 -542 0
// View Z:
viewdir 0 0 1
// View Y:
//viewdir 0 -1 0
lastlight
238
340.525269 -1410.304443 -99.83007 // Room corner 0
-125.449379 -1410.304443 -99.83007
340.525269 -0.267509 -99.829895
-125.449379 -0.267509 -99.829895
340.525269 -1410.304443 367.966248 // Room corner 4
340.525269 -0.267572 367.967529
-125.449379 -0.267572 367.967529
-125.449379 -1410.304443 367.966248 // Room corner 7
63.723488 -184.594345 -99.887794 // 8 Start of Sled base
63.723488 -876.187256 -99.887772
-61.389282 -184.594345 -99.887787 // 10
-61.389282 -876.187256 -99.887764
-80.434479 -895.917114 -99.887764
82.375648 -895.917114 -99.887772
-80.434479 -144.034225 -99.887787
82.375648 -144.034225 -99.887794
63.723488 -184.594315 -37.678402
82.375648 -144.034195 -37.678535
63.723488 -876.187256 -37.678375
82.375648 -895.917114 -37.678383
-61.389282 -876.187256 -37.678368 // 20
-80.434479 -895.917114 -37.678375
-80.434479 -144.034195 -37.678528
-61.389282 -184.594315 -37.678398 // 23 End of Sled base
-47.626122 -70.073471 367.929749  // light poles R1
-47.626122 -70.073456 176.710739
-50.676079 -70.073456 176.710739
-50.676079 -70.073471 367.929749
52.541649 -70.073471 367.929749
52.541649 -70.073456 176.710739
49.491676 -70.073456 176.710739 // 30
49.491676 -70.073471 367.929749
-47.288841 -204.722229 367.929749 // pair of light poles (8 vertices)
-47.288841 -204.722214 176.710739       // R2
-47.288841 -207.772171 176.710739
-47.288841 -207.772186 367.928711
49.491676 -207.772186 367.928711
49.491676 -207.772171 176.710739
49.491676 -204.722214 176.710739
49.491676 -204.722229 367.929749
-47.288841 -348.317444 367.929749 // 40 pair of light poles (8 vertices)
-47.288841 -348.317413 176.710739       // R3
-47.288841 -351.36734 176.710739
-47.288841 -351.367371 367.929749
49.491676 -351.367371 367.929749
49.491676 -351.36734 176.710739
49.491676 -348.317413 176.710739
```

49.491676 -348.317444 367.929749
-47.288841 -489.947052 367.930786 // pair of light poles (8 vertices)
-47.288841 -489.947021 176.710739 //v48-55 light poles R4
-47.288841 -492.996948 176.710739 // 50
-47.288841 -492.996979 367.929749
49.491676 -492.996979 367.929749
49.491676 -492.996948 176.710739
49.491676 -489.947021 176.710739
49.491676 -489.947052 367.930786
-47.288841 -635.16217 367.929749 // pair of light poles (8 vertices)
-47.288841 -635.16217 176.710739 //v56-59 light pole (near right)
-47.288841 -638.212097 176.710739 // R5
-47.288841 -638.212097 367.929749
49.491676 -638.212097 367.929749 // 60
49.491676 -638.212097 176.710739
49.491676 -635.16217 176.710739
49.491676 -635.16217 367.929749
-47.288841 -777.774597 367.929749 // pair of light poles (8 vertices)
-47.288841 -777.774597 176.710739 // R6
-47.288841 -780.824524 176.710739
-47.288841 -780.824524 367.929749
49.491676 -780.824524 367.929749
49.491676 -780.824524 176.710739
49.491676 -777.774597 176.710739 // 70
49.491676 -777.774597 367.929749
-50.676079 -918.421448 367.929749 // pair of light poles
-50.676079 -918.421448 176.710739 // R7
-47.626122 -918.421448 176.710739
-47.626122 -918.421448 367.929749
49.491676 -918.421448 367.929749
49.491676 -918.421448 176.710739
52.541649 -918.421448 176.710739
52.541649 -918.421448 367.929749
163.561157 -921.471375 367.928711 // 80 light poles L7 (8 verts)
163.561157 -921.471375 176.710739
163.561157 -918.421448 176.710739
163.561157 -918.421448 367.929749
263.728912 -921.471375 367.928711
263.728912 -921.471375 176.710739
263.728912 -918.421448 176.710739
263.728912 -918.421448 367.929749
163.561157 -638.442383 367.929688 // light poles L5 (8 verts)
163.561157 -638.442383 176.710739
163.561157 -635.392456 176.710739 // 90
163.561157 -635.392456 367.929749
263.728912 -638.442383 367.929688
263.728912 -638.442383 176.710739
263.728912 -635.392456 176.710739
263.728912 -635.392456 367.929749
163.561157 -493.979675 367.929688 // light poles L4 (8 verts)
163.561157 -493.979675 176.710739
163.561157 -490.929749 176.710739
163.561157 -490.929749 367.930725
263.728912 -493.979675 367.929688 // 100
263.728912 -493.979675 176.710739
263.728912 -490.929749 176.710739
263.728912 -490.929749 367.930725
163.561157 -351.482422 367.929688 // light poles L3 (8 verts)

163.561157 -351.482422 176.710739
163.561157 -348.432495 176.710739
163.561157 -348.432495 367.929749
263.728912 -351.482422 367.929688
263.728912 -351.482422 176.710739
263.728912 -348.432495 176.710739 // 110
263.728912 -348.432495 367.929749
163.561157 -208.985184 367.928711 // light poles L2 (8 verts)
163.561157 -208.985184 176.710739
163.561157 -205.935257 176.710739
163.561157 -205.935257 367.929749
263.728912 -208.985184 367.928711
263.728912 -208.985184 176.710739
263.728912 -205.935257 176.710739
263.728912 -205.935257 367.929749
163.561157 -70.173203 367.929688 // 120 light poles L1 (8 verts)
163.561157 -70.173203 176.710739
163.561157 -67.123276 176.710739
163.561157 -67.123276 367.929749
263.728912 -70.173203 367.929688
263.728912 -70.173203 176.710739
263.728912 -67.123276 176.710739
263.728912 -67.123276 367.929749
-125.463295 -258.760956 367.767334 // Joist 1 (8 verts)
-125.463303 -295.217621 367.767334
-125.463295 -258.760956 311.057007 // 130
-125.463303 -295.217621 311.057007
342.120544 -258.761017 367.767334
342.119781 -295.217682 311.057007
342.119781 -295.217682 367.767334
342.120544 -258.761017 311.057007
-125.463295 -1110.82959 367.767334 // Joist 4 (8 verts)
-125.463303 -1147.286133 367.767334
-125.463295 -1110.82959 311.057007
-125.463303 -1147.286133 311.057007
342.119812 -1110.82959 367.767334 // 140
342.119781 -1147.286133 311.057007
342.119781 -1147.286133 367.767334
342.119812 -1110.82959 311.057007
-125.463295 -862.9245 367.767334 // Dbl Joist far beam (8 verts)
-125.463295 -887.22876 367.767334
-125.463295 -862.9245 311.057007
-125.463295 -887.22876 311.057007
342.119812 -862.924561 367.767334
342.119812 -887.228882 311.057007
342.119812 -887.228882 367.767334 // 150
342.119812 -862.924561 311.057007
-125.463295 -811.075073 367.767334 // Dbl Joist near beam (8 verts)
-125.463295 -835.379333 367.767334
-125.463295 -811.075073 311.057007
-125.463295 -835.379333 311.057007
342.120544 -811.075134 367.767334
342.120544 -835.379456 311.057007
342.120544 -835.379456 367.767334
342.120544 -811.075134 311.057007
341.089813 -988.703979 336.17157 // 160 Sprinkler 4
-23.797316 -983.842957 336.17157
-23.7973 -988.703918 336.17157

341.089813 -983.843018 336.17157
341.090027 -706.772705 336.17157 // Sprinkler 3
-23.797752 -701.911682 336.17157
-23.797388 -706.772644 336.17157
341.090027 -701.911743 336.17157
-125.463295 -545.553162 367.767334 // Start Joist 2
-125.463318 -582.009766 367.767334
-125.463295 -545.553162 311.057007 // 170
-125.463318 -582.009766 311.057007
342.119812 -545.553223 367.767334
342.119781 -582.009888 311.057007
342.119781 -582.009888 367.767334
342.119812 -545.553223 311.057007 // End Joist 2
341.089325 -421.6008 336.17157 // Sprinkler 2
-23.797752 -416.739777 336.17157
-23.797739 -421.600739 336.17157
341.089325 -416.739838 336.17157
341.089325 -136.428955 336.17157 // 180 Sprinkler 1
-23.798103 -131.567932 336.17157
-23.798105 -136.428894 336.17157
341.089325 -131.567993 336.17157
333.433228 -8.163632 120.484055 // conduit
333.433228 -1403.400146 120.483917
-118.938965 -8.163632 120.48407
-118.938965 -1403.400146 120.483932
-125.169678 -1410.37207 120.483932
340.619812 -1410.37207 120.483932
-125.169678 -0.30853 120.484055 // 190
340.619812 -0.30853 120.484055
333.433228 -8.163635 132.109818
340.619812 -0.308533 132.109756
333.433228 -1403.400146 132.109665
340.619812 -1410.37207 132.109665
-118.938965 -1403.400146 132.109695
-125.169678 -1410.37207 132.109695
-125.169678 -0.308533 132.109756
-118.938965 -8.163635 132.109818 // end Conduit
32.52285 -135.585815 336.929626 // 200 Sprinkler 1 Hanger (8 verts)
29.472878 -135.585815 336.929626
29.472878 -132.535858 336.929626
32.52285 -132.535858 336.929626
32.52285 -135.586182 367.929443
29.472878 -135.586182 367.929443
29.472878 -132.536224 367.929749
32.52285 -132.536224 367.929749
32.52285 -420.772736 336.929626 // Sprinkler 2 Hanger (8 verts)
29.472878 -420.772736 336.929626
29.472878 -417.722778 336.929626 // 210
32.52285 -417.722778 336.929626
32.52285 -420.772827 367.929443
29.472878 -420.772827 367.929443
29.472878 -417.722839 367.929749
32.52285 -417.722839 367.929749
32.52285 -705.959717 336.929626 // Sprinkler 3 Hanger (8 verts)
29.472878 -705.959717 336.929626
29.472878 -702.90979 336.929626
32.52285 -702.90979 336.929626
32.52285 -705.959778 367.929443 // 220

```
29.472878 -705.959778 367.929443
29.472878 -702.90979 367.929749
32.52285 -702.90979 367.929749
32.52285 -987.905884 336.929626  // Sprinkler 4 hanger (8 verts) 224
29.472878 -987.905884 336.929626
29.472878 -984.855957 336.929626
32.52285 -984.855957 336.929626
32.52285 -987.905945 367.929443
29.472878 -987.905945 367.929443
29.472878 -984.855957 367.929749 // 230
32.52285 -984.855957 367.929749
-60.057274 -935.191345 162.016342 // 232-237 L Light 7
62.707253 -935.191345 162.016342
62.707253 -904.162964 176.856003
-60.057274 -904.162964 176.856003
-60.057274 -904.162964 162.016342
62.707253 -904.162964 162.016342 // 237
125 // =249 - 6 -118
4 237 236 235 234 0xfff both // lower visible side L Light 7
//3 237 235 234 0xfff both // upper visible side L Light 7
4 237 233 232 236 0xfff both // bottom far half L Light 7
//3 237 232 236 0xfff both // bottom near half L Light 7
//3 234 233 237 0xfff both
//3 231 230 229 0xfff both // Sprinker 4 hanger
//3 231 229 228 0x111 both
//3 231 227 226 0x111 both
//3 231 226 230 0x111 both
4 230 226 225 229 0x111 both
//3 230 225 229 0x111 both
4 229 225 224 228 0x111 both
//3 229 224 228 0x111 both
//3 228 224 227 0x111 both
//3 228 227 231 0x111 both
//3 227 224 225 0x111 both
//3 227 225 226 0x111 both
//3 223 222 221 0x111 both  // Sprinkler 3 Hanger
//3 223 221 220 0x111 both
//3 223 219 218 0x111 both
//3 223 218 222 0x111 both
4 222 218 217 221 0x111 both
//3 222 217 221 0x111 both
4 221 217 216 220 0x111 both
//3 221 216 220 0x111 both
//3 220 216 219 0x111 both
//3 220 219 223 0x111 both
//3 219 216 217 0x111 both
//3 219 217 218 0x111 both
//3 215 214 213 0x111 both  // Sprinkler 2 Hanger
//3 215 213 212 0x111 both
//3 215 211 210 0x111 both
//3 215 210 214 0x111 both
4 214 210 209 213 0x111 both
//3 214 209 213 0x111 both
4 213 209 208 212 0x111 both
//3 213 208 212 0x111 both
//3 212 208 211 0x111 both
//3 212 211 215 0x111 both
//3 211 208 209 0x111 both
```

//3 211 209 210 0x111 both
//3 207 206 205 0x111 both // Sprinkler Hanger 1
//3 207 205 204 0x111 both
//3 207 203 202 0x111 both
//3 207 202 206 0x111 both
4 206 202 201 205 0x111 both
//3 206 201 205 0x111 both
4 205 201 200 204 0x111 both
//3 205 200 204 0x111 both
//3 204 200 203 0x111 both
//3 204 203 207 0x111 both
//3 203 200 201 0x111 both
//3 203 201 202 0x111 both // end of sprinkler hanger 1
3 199 198 197 0xaaa both // Conduit Start
3 196 199 197 0xaaa both
3 196 197 195 0xaaa both
3 194 196 195 0xaaa both
3 194 195 193 0xaaa both
3 192 194 193 0xaaa both
3 193 198 199 0xaaa both
3 193 199 192 0xaaa both
3 198 193 191 0xaaa both
3 198 191 190 0xaaa both
3 193 195 189 0xaaa both
3 193 189 191 0xaaa both
3 195 197 188 0xaaa both
3 195 188 189 0xaaa both
3 197 198 190 0xaaa both
3 197 190 188 0xaaa both
3 199 196 187 0xaaa both
3 199 187 186 0xaaa both
3 196 194 185 0xaaa both
3 196 185 187 0xaaa both
3 194 192 184 0xaaa both
3 194 184 185 0xaaa both
3 192 199 186 0xaaa both
3 192 186 184 0xaaa both
3 188 190 186 0xaaa both
3 188 186 187 0xaaa both
3 189 188 187 0xaaa both
3 189 187 185 0xaaa both
3 191 189 185 0xaaa both
3 191 185 184 0xaaa both
3 186 190 191 0xaaa both
3 184 186 191 0xaaa both
//3 183 182 181 0x111 both // Sprinkler 1
4 183 180 182 181 0x111 both
//3 179 178 177 0x111 both // Sprinkler 2
4 179 176 178 177 0x111 both
//3 175 174 173 0xfff both // Joist 2 end
//3 175 172 174 0xfff both
//3 175 171 170 0xaaa both // Joist 2 bottom
4 175 173 171 170 0xaaa both // Joist 2 bottom
//3 173 169 171 0x888 both // Joist 2 far side
4 173 174 169 171 0x888 both // Joist 2 far side
//3 174 168 169 0xfff // Joist 2 top
//3 174 172 168 0xfff // Joist 2 top
//3 172 170 168 0x888 both // Joist 2 near side

```
4 172 175 170 168 0x888 both // Joist 2 near side
//3 170 169 168 0xfff both // Joist 2 end
//3 170 171 169 0xfff both
//3 167 166 165 0x111 both // Sprinkler 3
4 167 164 166 165 0x111 both
//3 163 162 161 0x111 both  // Sprinkler 4
4 163 160 162 161 0x111 both
//3 159 158 157 0xfff both // Dbl Joist A end
//3 159 156 158 0xfff both
//3 159 155 154 0xaaa both // Dbl Joist A bottom
4 159 157 155 154 0xaaa both // Dbl Joist A bottom (50 rendered polys here)
//3 157 153 155 0xfff both // Dbl Joist A far side
4 157 158 153 155 0xfff both
//3 158 152 153 0xfff both // Dbl Joist A top
//3 158 156 152 0xfff both
//3 156 154 152 0x888 both // Dbl Joist A near side
4 156 159 154 152 0x888 both // Dbl Joist A near side
//3 154 153 152 0xfff both // Dbl Joist A end
//3 154 155 153 0xfff both
//3 151 150 149 0xfff both // Dbl Joist B end
//3 151 148 150 0xfff both
//3 151 147 146 0xaaa both // Dbl Joist B bottom
4 151 149 147 146 0xaaa both // Dbl Joist B bottom
//3 149 145 147 0xfff both // Dbl Joist B far side
4 149 150 145 147 0xfff both
//3 150 144 145 0xfff both  // Dbl Joist B top
//3 150 148 144 0xfff both
//3 148 146 144 0x888 both // Dbl Joist B near side
4 148 151 146 144 0x888 both // Dbl Joist B near side
//3 146 145 144 0xfff both // Dbl Joist B end
//3 146 147 145 0xfff both
//3 143 142 141 0xfff both  // Joist 4 end
//3 143 140 142 0xfff both
//3 143 139 138 0xaaa both // Joist 4 bottom
4 143 141 139 138 0xaaa both // Joist 4 bottom
//3 141 137 139 0xfff both  // Joist 4 far side
4 141 142 137 139 0xfff both
//3 142 136 137 0xfff both          // Joist 4 top
//3 142 140 136 0xfff both
//3 140 138 136 0x888 both // Joist 4 near side
4 140 143 138 136 0x888 both // Joist 4 near side
//3 138 137 136 0xfff both // Joist 4 end
//3 138 139 137 0xfff both //
//3 135 134 133 0xfff both // Joist 1 end
//3 135 132 134 0xfff both // Joist 1 end
//3 135 131 130 0xaaa both // Joist 1 bottom near half
4 135 133 131 130 0xaaa both // Joist 1 bottom far half
//3 133 129 131 0x888 both // Joist 1 far side lower half
4 133 134 129 131 0x888 both // Joist 1 far side upper half
//3 134 128 129 0xfff both // Joist 1 top
//3 134 132 128 0xfff both // Joist 1 top
//3 132 130 128 0x888 both // Joist 1 near side upper half
4 132 135 130 128 0x888 both // Joist 1 near side lower half
//3 130 129 128 0xfff both // Joist 1 end
//3 130 131 129 0xfff both // Joist 1 end
4 127 126 125 124 0xfff both //     Light poles L1
//3 127 125 124 0xfff both
4 123 122 121 120 0xfff both
```

```
//3 123 121 120 0xfff both
4 119 118 117 116 0xfff both // Light pole 2 L left
//3 119 117 116 0xfff both // Light pole 2 L left
4 115 114 113 112 0xfff both
//3 115 113 112 0xfff both
4 111 110 109 108 0xfff both
//3 111 109 108 0xfff both
4 107 106 105 104 0xfff both
//3 107 105 104 0xfff both
4 103 102 101 100 0xfff both
//3 103 101 100 0xfff both
4 99 98 97 96 0xfff both // Light pole 4 L right
//3 99 97 96 0xfff both // Light pole 5 L right
4 95 94 93 92 0xfff both // Light pole 5 L left
//3 95 93 92 0xfff both // Light pole 5 L left
4 91 90 89 88 0xfff both
//3 91 89 88 0xfff both
4 87 86 85 84 0xfff both
//3 87 85 84 0xfff both
4 83 82 81 80 0xfff both
//3 83 81 80 0xfff both
4 79 78 77 76 0xfff both
//3 79 77 76 0xfff both
4 75 74 73 72 0xfff both
//3 75 73 72 0xfff both
4 71 70 69 68 0xfff both
//3 71 69 68 0xfff both
4 67 66 65 64 0xfff both
//3 67 65 64 0xfff both
4 63 62 61 60 0xfff both
//3 63 61 60 0xfff both
4 59 58 57 56 0xfff both
//3 59 57 56 0xfff both
4 55 54 53 52 0xfff both
//3 55 53 52 0xfff both
4 51 50 49 48 0xfff both
//3 51 49 48 0xfff both
4 47 46 45 44 0xfff both
//3 47 45 44 0xfff both
4 43 42 41 40 0xfff both
//3 43 41 40 0xfff both
4 39 38 37 36 0xfff both
//3 39 37 36 0xfff both
4 35 34 33 32 0xfff both // Light pole 2 R right
//3 35 33 32 0xfff both // Light pole 2 R right
4 31 30 29 28 0xfff both // Light pole 1 R left
//3 31 29 28 0xfff both // Light pole 1 R left
4 27 26 25 24 0xfff both // Light pole 1 R right
//3 27 25 24 0xfff both // Light pole 1 R right
3 23 22 21 0xaaa both // start of aaa color Sled base
3 20 23 21 0xaaa both
3 20 21 19 0xaaa both // tried
3 18 20 19 0xaaa both
3 18 19 17 0xaaa both
3 16 18 17 0xaaa both
3 17 22 23 0xaaa both
3 17 23 16 0xaaa both
3 22 17 15 0xaaa both
```

```
3 22 15 14 0xaaa both
3 17 19 13 0xaaa both // tried
3 17 13 15 0xaaa both
3 19 21 12 0xaaa both // 100 rendered polys here
3 19 12 13 0xaaa both
3 21 22 14 0xaaa both // tried
3 21 14 12 0xaaa both // tried
3 23 20 11 0xaaa both // sled base R inner upper
3 23 11 10 0xaaa both // sled base R inner lower
3 20 18 9 0xaaa both // tried
3 20 9 11 0xaaa both // sled base far end inner lower
3 18 16 8 0xaaa both // sled base L inner upper
3 18 8 9 0xaaa both // sled base L inner lower
3 16 23 10 0xaaa both // tried
3 16 10 8 0xaaa both // tried
3 12 14 10 0xaaa both
3 12 10 11 0xaaa both
3 13 12 11 0xaaa both
3 13 11 9 0xaaa both
3 15 13 9 0xaaa both
3 15 9 8 0xaaa both
3 10 14 15 0xaaa both
3 8 10 15 0xaaa both
4 7 6 5 4 0xccc both // roof right half
//3 4 7 5 0xccc both // roof left half
4 3 2 5 6 0xbbb both // near end lower half
//3 6 3 5 0xbbb both // near end upper half
4 1 3 6 7 0xddd both // right wall lower half
//3 7 1 6 0xddd both // right wall upper half
4 0 1 7 4 0xbbb both // far end lower half
//3 4 0 7 0xbbb both // far end upper half
4 2 0 4 5 0xddd both // left wall lower half
//3 5 2 4 0xddd both // left wall upper half
4 1 0 2 3 0xeee // floor far end
//3 3 1 2 0xeee // floor near end
// sldrmz5.nff END
```

Configuration File "sledz.cfg"

```
scenes\
profiles\
sled\
sledz2.txt
```

Experiment Description File "sledz2.txt"

12	Number of trials
hmd32.txt	Scene motion file
sldrmz5.nff	Scene descrption file
nodata.dat	Save data file (not used in this experiment)
1	Trial number
30	Expected FPS (not used in this experiment. See Appendix A)
2	Subject posture (not used in this experiment)
1	Angular velocity direction (cw/ccw) (not used in this experiment)
hmd32.txt	Next trial

```
sldrmz5.nff
2
30
2
1
hmd12.txt
sldrmz5.nff
3
30
2
1
hmd12.txt
sldrmz5.nff
4
30
2
1
hmd22.txt
sldrmz5.nff
5
30
2
1
hmd22.txt
sldrmz5.nff
6
30
2
1
hmd12.txt
sldrmz5.nff
7
30
2
1
hmd32.txt
sldrmz5.nff
8
```

30
2
1
hmd22.txt
sldrmz5.nff
9
30
2
1
hmd22.txt
sldrmz5.nff
10
30
2
1
hmd32.txt
sldrmz5.nff
11
30
2
1
hmd12.txt
sldrmz5.nff
12
30
2
1

Appendix D Data Acquisition Software

All data acquisition was performed using Labtech Control software, from Labtech Corp. (Wilmington, MA). The software is icon-based. This appendix consists of a display of our iconified acquistion structure.

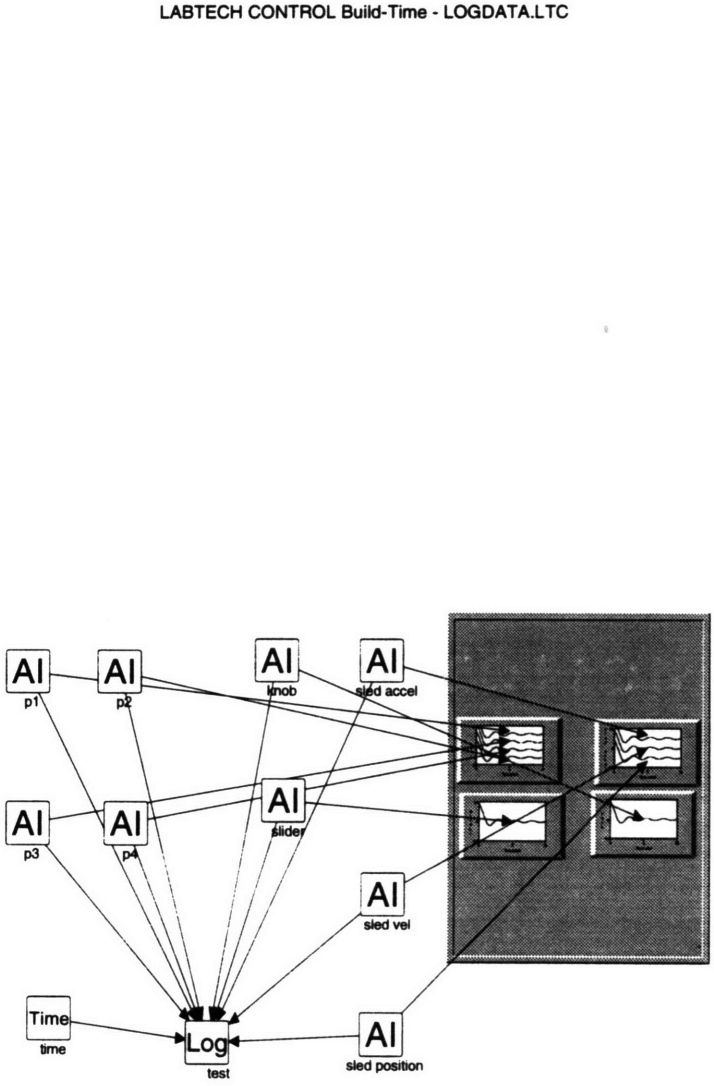


Figure D-1 Iconified data acquisition program

References

Albery, W.B., and Hunter, E.D. (1978) "G-Seat Component Development" AFHRL-TR-78-18.

Ashworth, B. R., McKissick, B. T., and Parrish, R. V. (1984) "Effects of Motion Base and g-Seat Cueing on Simulator Pilot Performance" NASA Technical Paper 2247.

Berthoz A., Pavard B., Young L.R. (1975) "Perception of Linear Horizontal Self-Motion induced by Peripheral Vision, Basic Characteristics and Visual-Vestibular Interactions" Exp. Brain Res 23, 471-489.

Borah, J., Young, L.R. and Curry, R.E. (1978) "Sensory Mechanism Modeling" AFHRL-TR-78-83.

Brandt, T., Dichgans, J. and Koenig, E. (1973) "Differential Effect of Central Versus Peripheral Vision on Egocentric and Exocentric Motion Perception". Exp. Brain Res. 16, 476-491.

Christie, J.R.I. (1992) "Modulation of Optokinetic Nystagmus in Humans by Linear Acceleration, and the Effects of Space Flight" M.S. Thesis, Massachusetts Institute of Technology, 1992.

Fernandéz and Goldberg (1976) "Physiology of peripheral neurons innervating otolith organs of the squirrel monkey". J. Neurophysiol. 39: 970-1008.

Fisher, M. H. and Kornmüller, A. E. (1931) "Optokinetisch ausgelöste Bewegungswahrnehmungen und optokinetischer Nystagmus". J. Phychol. Neurol. (Lpz.) 41, 383-420.

DeSouza, J. (1995) "A Virtual Environment System for Spatial Orientation Research" M.S. Thesis, Massachusetts Institute of Technology.

Helmholtz, H. (1910) Handbuch der physiologischen Optik, Vol III, 3.: L. Voss, Hamburg-Leipzig.

Hiltner, D.W. (1983) "A Closed-Loop Otolith System Assessment Procedure" M.S. Thesis, Massachusetts Institute of Technology.

Horch, K.W., Tucket, R.P. and Burgess, P.R. (1977) "A Key to the Classification of Cutaneous Mechanoreceptors". Journal of Investigative Dermatology 69: 75-82.

Israël, I., Chapuis, N., Glasauer, S., Charade, O., and Berthoz, A. (1993) "Estimation of passive Horizontal Linear Whole-Body Displacement in Humans." Journal of Neurophysiology 70: 1270-1273.

Kleinwaks, J. M. (1980) "Advanced Low Cost G-Cuing System". AFHRL-TR-79-62.

Kurtz, K.J. and Warren, W.H. (1992) " The Role of Central and Peripheral Vision in Perceiving the Direction of Self-Motion". Perception & Psychophysics, 51 (5), 443-454.

Lichtenberg, B.K. (1979) "Ocular Counterrolling Induced in Humans By Horizontal Accelerations." Sc.D. Thesis, Massachusetts Institute of Technology.

Loewenstein, W.R. (1971) "Mechano-electric Transduction in the Pacinian Corpuscle Initiation of Sensory Impulses in Mechanoreceptors". Handbook of Sensory Physiology Vol.1 Principles of Receptor Physiology . Springer-Verlag, Berlin.

Meiry, J .L. (1965) "The Vestibular System and Human Dynamic Space Orientation". Sc.D. Thesis, Massachusetts Institute of Technology.

Melvill Jones, G. and Young, L. R. (1978) "Subjective Detection of Vertical Acceleration: A Velocity-Dependent Response?" Acta Otolaryngol 85:45-53.

Postman, L. and Egan, J. (1949) Experimental psychology: An Introduction. Harper, New York.

Rosner, B. (1995) Fundamentals of Biostatistics, Fourth Edition Wadsworth Publishing Co., Belmont, CA USA.

Shirley, R.S. and Young, L.R. (1968) "Motion Cues in Man-Vehicle Control Effects of Roll-Motion Cues on Human Operator's Behavior in Compensatory Systems with Disturbance Inputs". IEEE Transactions on Man-Machine System, Vol. MMS-9 , No.4.

Skwersky, A. (1996) "Effect of Scene Polarity and Head Orientation on Roll Illusions in a Virtual Environment." M.S. Thesis, Massachusetts Institute of Technology.

Warren, W. H. and Kurtz, K. J. (1979) "The role of central and peripheral vision in perceiving the direction of self-motion." Perception and Psychophysics 51 (5):443-454.

Young, L. R. (1979) "Visual-Vestibular Interaction". Posture and Movement, Raven Press, New York.

Young, L. R. (1984) "Perception of the Body in Space: Mechanisms". Handbook of Physiology- The Nervous System III vol.3, part 2, Waverly Press, Baltimore, MD

Young, L. R. and Meiry, J. L. (1965) "Bang-Bang Aspects of Manual Control in High-Order Systems" IEEE Transactions on Automatic Control. Volume AC-10, Number 3, July, 1965 pp. 336-341.

Young L. R. and Meiry, J. L. (1968) "A Revised Dynamic Otolith Model" Aerospace Medicine 39: 606-608.

Young, L.R. and Shelhamer, M. (1990) "Microgravity Enhances the Relative Contribution of Visually-Induced Motion Sensation". Aviation, Space and Environmental Medicine, Aerospace Medical Association, Washington DC.

Young, L. R., Meiry, J. L. and Li, Y. T. (1966) "Control Engineering Approaches to Human Dynamic Space Orientation". Second Symposium on the Role of the Vestibular Organs in Space Exploration, Nation Aeronautics and Space Administration, Ames Research Center, Moffett Field, CA. NASA SP-115.

Weber, E. H.(1834) Der Tactu (H.E. Ross, trans.) Academic, New York (1978)

Zecharias, G. L. (1977) "Motion Sensation Dependence on Visual and Vestibular Cues" Ph.D. Massachusetts Institute of Technology, 1977.