

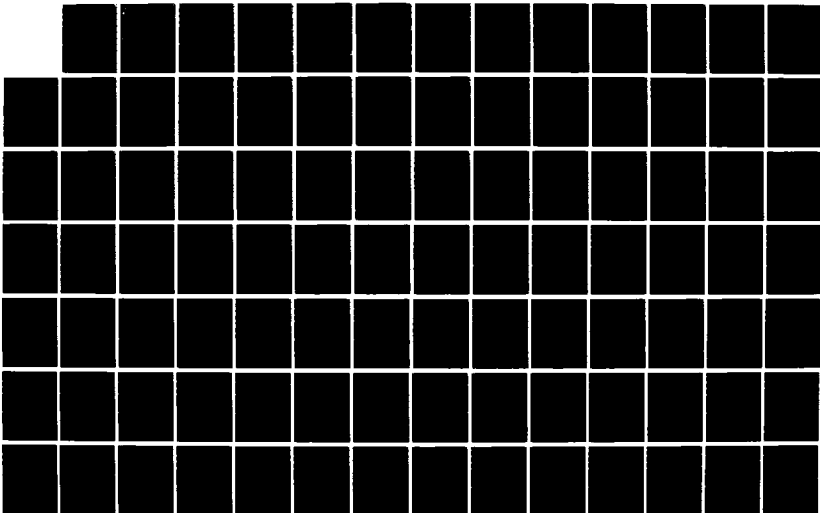
AD-A189 674

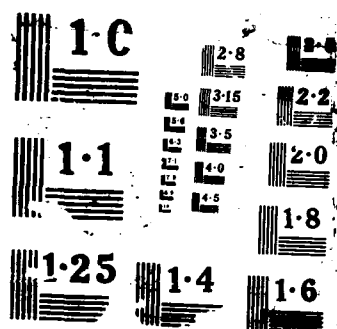
MOTION SICKNESS: QUANTITATIVE ALGORITHMIC MALAISE
INDICATION IN REAL TIME(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING E L FIX
DEC 87 AFIT/GE/ENG/87D-18 F/G 6/10

1/2

UNCLASSIFIED

NL

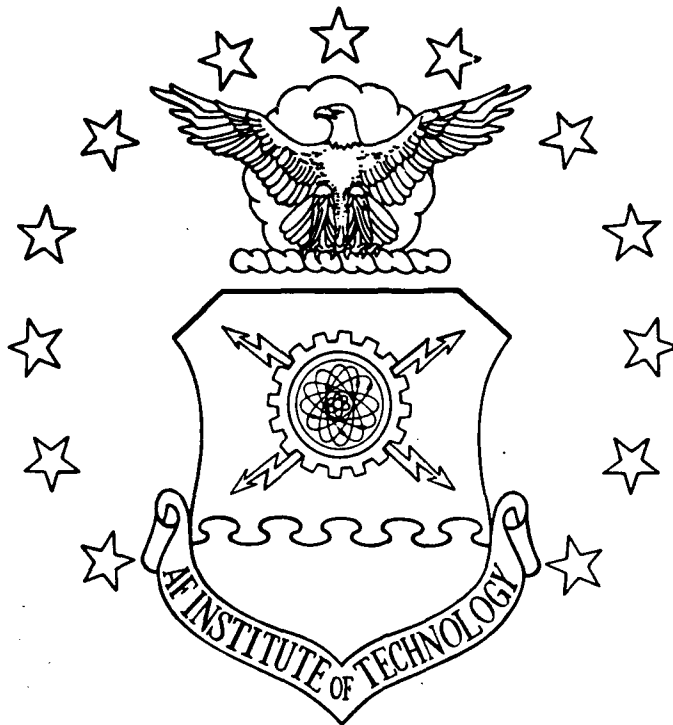




DTIC FILE COPY

1

AD-A189 674



MOTION SICKNESS: QUANTITATIVE,
ALGORITHMIC MALAISE INDICATION
IN REAL TIME

THESIS

Edward L. Fix
Captain, USAF

AFIT/GE/ENG/87D-18

DTIC
ELECTE
MAR 07 1988

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

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

02 3 01 090

①
AFIT/GE/ENG/87D-18

MOTION SICKNESS: QUANTITATIVE,
ALGORITHMIC MALAISE INDICATION
IN REAL TIME

THESIS

Edward L. Fix
Captain, USAF

AFIT/GE/ENG/87D-18

DTIC
SELECTED
MAR 07 1988
S H

Approved for public release; distribution unlimited

AFIT/GE/ENG/87D-18

MOTION SICKNESS: QUANTITATIVE, ALGORITHMIC
MALAISE INDICATION IN REAL TIME

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Edward L. Fix, B.S.
Captain, USAF

December 1987

Approved for public release; distribution unlimited

Preface

This thesis research was a team effort with Capts Drylie and Gaudreault. The first two chapters are nearly identical in all three theses, but we avoided duplication in the remainder. Therefore, the reader needs all three theses to completely understand the experiment and results.

The goal of this effort was to gather data on subjects as they became motion sick and analyze it. We hoped proper analysis would reveal a pattern of how motion sickness evolves, and provide insight about the mechanisms involved. We wanted to develop a real time model which could read physiological signals from the subject and compute his level of discomfort for use in biofeedback training.

I am grateful to many people who contributed to this effort in many ways. Dr. Kabrisky was an unfailing source of encouragement, enthusiasm, constructive criticism and good ideas. Dr. Bill Czelen's knowledge and unique analytical mind were indispensable. Mr. Bob Durham, Capt Clifford and Mr. Jim Ater provided vital support. My colleagues, Capts Drylie and Gaudreault, really made it possible.

Finally, I would like to thank my wife, Jan, for her support and understanding all the days and nights when all she saw of me was the back of my head as I worked at my desk. She provided the moral support when I needed it.

Edward L. Fix

Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

1010148M
A402
B118

ion For
GRA&I ☒
8 ☐
need ☐
ation

Table of Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	1
Summary of Current Knowledge	4
Problem	6
Assumptions	6
Scope	7
Materials, Equipment, and Software	7
Other Support	9
II. Experimental Procedure	10
Environment	10
Procedures	11
Sensors	11
Recording and Processing Equipment	13
III. A Mathematical Model of Motion Sickness	14
Data Calibration	14
Data Preprocessing	16
Curve Fitting	18
The Composite Equation	20
IV. Real Time Implementation of the Model	26
Specialized Equipment and Software	26
Adaptation of the Model	27
V. Neural Network Simulation	33
Single Node Perceptrons	33
Multi-Layer Perceptrons	34
Perceptron Simulation	36
VI. The Road Ahead	42
Summary and Conclusions	42
Recommendations	44

	Page
Appendix A: Equation and Neural Net Error Signals .	46
Appendix B: Motion Sickness Indicator Program Operating Instructions	51
Appendix C: Motion Sickness Indicator Program C Source Code	54
Appendix D: Neural Network Simulation Program C Source Code	77
Appendix E: Neural Net Real Time Implementation C Source Code	82
Appendix F: Neural Net Weights	86
Appendix G: Neural Net Training Data Set	90
Bibliography	97
Vita	99

List of Figures

Figure	Page
1. Electrosplanchnogram Signal	17
2. RMS Electrosplanchnogram Signal	18
3. Computed and Reported Indices	23
4. Equation Error	24
5. Output Screen Display	32
6. Single Neuromime Element	34
7. Neural Network	35
8. Net Error Signal	39
9. Error Signals Subject 1	47
10. Error Signals Subject 2	48
11. Error Signals Subject 3	49
12. Error Signals Subject 4	50

List of Tables

Table	Page
1. Intermediate Linear Coefficient Results	22
2. Equation Degraded Performance	25
3. Net Degraded Performance	40
4. Average Absolute Errors	44
5. Subject 1 Degraded Performance Average Absolute Error	44

Abstract

→ Physiological data were collected on human volunteers to study the effects of motion sickness. Data were analyzed and correlated using least squares curve fitting and other statistical methods that are described. An equation is developed that relates five separate physiological signals to subjective motion sickness.

A computer program is presented and described which takes the physiological signals in real time and computes the motion sickness. A pattern recognition type of approach which uses a neural net is presented and discussed as an alternative to the equation model. The two models are compared.

→ Summary 1

MOTION SICKNESS: QUANTITATIVE, ALGORITHMIC
MALAISE INDICATION IN REAL TIME

I. Introduction

Motion sickness is a perplexing and expensive problem for the military services and NASA. A trained aircrew member who must be grounded for air sickness represents a substantial loss (12; 17). Both the money spent training the individual and the time to train a replacement are lost.

It has been a problem for astronauts as well. Flight schedules are often disrupted to accommodate the problem (11). Finding the causes and a cure for motion sickness could save a great deal of money and promote smoother operations.

This disorder is characterized by a variety of symptoms; the most prevalent are nausea, pallor, sweating, and vomiting. Other possible symptoms include salivation, feeling of warmth, light-headedness, depression or apathy, yawning and drowsiness, belching or flatulence, headache, and occasionally hyperventilation. They are brought on by unusual or provocative motion stimulus, either real or perceived (3:469).

The leading theory about the mechanism of motion sickness is the sensory conflict theory. It says that when there is a conflict between different parts of the balance system, motion sickness can result (3:474-481). The balance

reflex uses the information from several different senses; primarily the vestibular, or inner ear, and the visual system. There is also input from the somatic senses which report the position of body parts and pressure on various surfaces (7:309-310, 323). If the brain perceives these various signals to be in conflict compared to the normal motion cues, motion sickness can result.

There have been two primary avenues of motion sickness treatment: drugs and biofeedback. Each has been somewhat successful in different circumstances.

→ Drug treatment is the easiest method to apply for alleviating motion sickness; however, drugs have undesirable side effects. In fact, aviators flying solo are prohibited from taking anti-motion-sickness drugs (3:491).

→ Biofeedback is a promising treatment method for long-term protection. The School of Aerospace Medicine (SAM) at Brooks MC has had success using skin resistance, muscle tension, and skin temperature as feedback parameters under coriolis stress. Of the aircrew members treated this way, about 84% in one study were returned to flying status (12:119-121). → A problem with biofeedback, however, is identifying physiologic parameters that are good indicators of motion sickness and can also be brought under voluntary control.

Ashton Graybiel and his team took a first step by developing a standard scale for measuring motion sickness

discomfort (8). Others have measured skin potential and resistance and correlated them with discomfort (11:141). This provides the basis for selecting parameters for biofeedback treatment, leading to recent AFIT thesis efforts.

In 1983, Capts Earl and Peterson first performed AFIT motion sickness research sponsored by SAM to develop a biofeedback system. Their effort, carried on by Fitzpatrick, Rogers, and Williams in 1984 and by Jarvis and Uyeda in 1985, involved automating data collection and biofeedback parameter presentation so the work formerly performed at SAM could be performed at other bases to reduce cost (5). This effort ultimately proved unsuccessful due to problems with computer equipment. During the course of the experiments, however, the researchers learned a great deal about gathering and analyzing biophysical data and constructed several physiologic sensors to gather a wide range of data (5,12). In 1984 Fitzpatrick, Rogers, and Williams suggested building an automated motion sickness prediction model (5:6-1). This has been the thrust of more recent thesis efforts at AFIT (9; 15; 16).

In 1986, Hartle, McPherson, and Miller began integrating the subject's report of discomfort with the other data and correlated the reports with Graybiel's model (9:53-54; 8). Using automated statistical methods, they were able to establish positive relationships between

several physiological parameters and the reported motion sickness index (9:55). This led to development of equations relating several parameters to motion sickness (9:97, 15:59, 16:84).

Summary of Current Knowledge

Several important pathological findings also came out of the 1986 research effort in the areas of galvanic skin response, electroencephalogram, heart rate, skin pallor, respiration, and gastrointestinal activity.

Galvanic Skin Response. Other work in the field has shown a positive relationship between nociceptive stimuli and increasing skin conductance (18:420,421). This was confirmed by the AFIT work (15:37).

Gastro-Intestinal Activity. The AFIT researchers computed the power spectral densities of the stomach (EGG) and intestinal (ESG) signals. They found that the signals can be separated on the basis of this type of analysis (9:108).

Electroencephalogram. One of the most exciting discoveries in 1986 was the existence of extremely high amplitude (1 - 5 millivolts), low frequency (0.1 - 0.2 Hertz) waves (2). This previously unknown phenomenon may hold a key to understanding motion sickness. According to Dr. William Czelen (2), EEG responses of these amplitudes but at higher frequencies have been seen during tonic-clonic seizures, hypercarbia, and asphyxia. It is possible that

anticonvulsant drugs may prevent or lessen nausea. This possibility has tremendous implications for aircrew rehabilitation and space flight.

Heart Rate. Although subjects were instructed to halt the experiment just before emesis, one subject inadvertently continued through emesis. He was one of a group of subjects who demonstrated sinus arrest, i.e., the vagal pacemaker signal to the heart was apparently temporarily blocked and the heart rhythm converted to a junctional or ventricular one at a slower rate (35 - 40 beats per minute). When the subject vomited, his heart rate nearly tripled immediately (16:66- 67), converting to a sinus tachycardia. All subjects showed increasing heart rate until reporting quite high symptom levels, and then decreasing heart rate until just before emesis. Several later subjects who volunteered to continue to emesis showed the same increase after emesis, although the sinus arrest condition did not necessarily occur.

Skin Pallor. Jarvis and Uyeda found skin pallor patterns linked to motion sickness in 1985 (10:87-88). Hartle, McPherson, and Miller also found pallor changes in 1986, but their data seem to suggest the face flushes while the hands become more pale (15:39). This may have been due to sensor placement problems and unrelated to actual blood flow (2). The problem remained to be resolved at the conclusion of their work.

Respiration. Hartle, McPherson, and Miller found that "the number of breaths increased by approximately 20 percent . . . and the respiratory contribution from the diaphragm increased by about 50 percent" (16:68-69).

Electrosplanchnogram. Hartle, McPherson, and Miller found that the electrointestinogram signal "increased by almost 500 percent" (16:73) and that the electrogastrogram signal "frequency shifts from .12 hz to .06 hz" (16:77).

Problem

The purpose of the present study was to collect and analyze biophysical data relating to motion sickness, test the accuracy and improve upon motion sickness predictors developed by previous AFIT thesis teams, and develop a real-time processor to predict a subject's level of discomfort.

Assumptions

The assumptions for this research effort are:

1. Motion sickness induced in the laboratory is the same disorder as that found in the real world.
2. The physiological data have a definite correlation to the degree of motion sickness.
3. Biofeedback techniques developed from a statistical analysis of physiological signals can be used to control motion sickness.

Scope

This is a follow-on research effort to continue the study of motion sickness at AFIT by performing experiments

on more subjects and improving methods of data analysis. This research team experimented with changes in the data analysis equipment. The scope of this research was limited to:

1. Collecting new motion sickness data on 25 volunteers.
2. Standardizing the test procedures.
3. Improving those sensors that proved unreliable in the past.
4. Testing and improving the mathematical models already developed to predict the subjective degree of motion sickness.
5. Integrating new equipment and software into the experimental procedures including:
 - a. A differential stethoscope for monitoring gastro-intestinal sounds.
 - b. A 16-channel bank of low pass filters to reduce electrical noise.
 - c. A 16-channel strip chart recorder.
 - d. Data acquisition and analysis software for the Zenith 248 computer.

Materials, Equipment, and Software

Materials. Materials include disposable electrodes, alcohol cleaning pads, Beta format video tapes, diskettes, Subject Questionnaires and Histories, and 16-channel thermal strip chart paper.

Equipment. The equipment included the following:

1. The powered rotating chair with the following physiological sensors constructed by Dr. Czelen:
 - a. electrocardiograph.

- b. two thermistors to measure skin surface temperature.
 - c. two electronystagmographs to measure eye movement.
 - d. galvanic skin reflex sensor to measure skin resistivity.
 - e. two photo-plethysmographs to measure pallor.
 - f. two electrosplanchnographs to measure gastric and intestinal electrical skin surface potentials.
 - g. two pneumographs to measure respiration.
 - h. ballistocardiograph to measure cardiac induced thoracic oscillation.
 - i. three electroencephalographs to measure brain wave activity.
2. The Zenith Z-248 personnel computer with peripheral units including an 8-channel analog-to-digital converter, and waveform scroller.
 3. The Marshall Electronics' Astropulse 90 sphygmomanometer.
 4. The SOLTEC model 8K20 series 16-channel strip chart recorder.
 5. The Kyowa Dengyo 14-channel Beta tape recorder.
 6. The AMPEX FR 1300 16-track FM tape recorder.
 7. The 16-channel low pass filter bank constructed by Dr. Czelen.
 8. The INTECH Systems' DIF-STET differential stethoscope.
 9. The Spiropet pocket Spirometer.
 10. The Cyborg Thermal P642 digital thermometer.

Many of the measurements that were made were only possible because of the equipment that Dr. Czelen designed and built.

Software. The software included packages for both data analysis and acquisition. Commercial software packages included DATAQ Instruments' Cudas for digitizing and displaying wave forms, MacMillan Software's Asystant for numerical and statistical analysis, and Metrabyte's C Tool for driving an analog-to-digital converter. The researchers developed software to calculate and report the subject's degree of motion sickness in real time.

Other Support

Dr. William Czelen (M.D., B.S.E.E.) provided necessary medical expertise for screening volunteers and ensuring the physical well-being of all subjects. Also, he provided technical support through circuit design and fabrication.

II. Experimental Procedure

This experiment was a continuation of work performed in previous years by Hartle, McPherson, and Miller, by Jarvis and Uyeda, and others (5; 9; 10; 15; 16). The experimental procedure consisted of spinning a volunteer subject about his Z axis in a powered rotating chair at a constant controlled speed, and gathering up to 21 channels of physiological data as the subject tilted his head out of the plane of rotation to elicit a motion sickness response. The subject reported his subjective discomfort on a numerical score from 1 to 10, where 1 meant the subject was asymptomatic and 10 meant emesis was imminent. The data were then analyzed by appropriate statistical techniques to determine relationships between the parameters and a computer model was developed to compute in real time an indication of the subject's numerical score. More detailed descriptions are available in the previous theses and in Gaudreault (6:14-21). Following are the major changes from the previous experiments.

Environment

The motion chair is presently installed in an unair conditioned environment. When experiments were conducted with high ambient temperatures, thermal sweating regularly loosened the electrodes and sensors from the subject's skin. Partitions were set up surrounding the motion chair and a

parachute was suspended above it to enclose the area. A portable room air conditioner was installed to help control the environment immediately around the chair. In this way, it was sometimes possible to maintain the temperature in the ideal 22 to 24° C. (71 to 75° F.) range (18:418) when the ambient temperature would otherwise have prevented experimental runs.

Procedures

The order of preparation of the subject was changed from the procedures the previous thesis team used to minimize the amount of time the electrodes were attached to the subjects' skin. The physical exam was done first, then the plethysmographs were calibrated. The body electrodes and sensors were attached before the subject mounted the chair, and the leads to the sensor electronics were connected afterwards. The pneumographs were then calibrated, and the facial electrodes and sensors were attached. Finally, the eyes were taped closed and the subdermal EEG electrodes were inserted. This method minimized sensor losses caused by perspiration loosening the adhesive.

Sensors

EEG. The previous thesis team discovered extremely low frequency, high amplitude EEG signals associated with motion sickness. To eliminate the possibility that these signals

were a sweat induced artifact, subdermal electrodes are now used.

Skin Pallor. The plethysmographs used before had a red LED and a photo transistor to measure the change in skin reflectivity due to flushing. However, flushed skin looks darker because the hemoglobin absorbs shorter wavelengths of light and reflects only red. Thus, the reflectivity of skin in red light changes very little, but the reflectivity in other colors (like green) changes dramatically. The LEDs have therefore been changed to green to take advantage of this effect. Also a second phototransistor, covered with opaque paint and connected to a balancing circuit, has been added to each sensor for temperature compensation. Both sensors were moved to the sub-orbital area of the face, one on each side. This area shows significant pallor change and has no large blood vessels. Finally, the adhesive used to hold the sensors in place irritated the skin of many subjects, masking any skin color change due to blood flow change. The plethysmographs are now held in place by taping over the top of the sensor. Because of these problems, the skin pallor data gathered before these changes have not been used in this study.

Gastro-Intestinal Measurements. A phonosplachnogram has been added to the data collected. The sensor is a battery powered, self contained stethoscope with differential inputs. It is attached to the central

abdominal region and records bowel sound activity. The output is recorded with the FM data recorder.

Recording and Processing Equipment

The previously used Bushmark strip chart recorders have been replaced by a Soltec model 8k26 16 channel chart recorder. It uses heat sensitive paper rather than ink, and is much easier to operate than the old recorders.

A Zenith 248 computer with an 8 channel analog-to-digital converter board was used to digitize data and analyze it. The computer was also used to analyze the data in real time and could be used to provide biofeedback information.

The previous researchers had problems with 60 Hz hum in their data. To solve the problem, a 16 channel active filter bank has been added to the data circuit. It uses active single pole low pass filters with 3dB points set at 30 Hz. These filters eliminated all significant 60 Hz noise from the recorded data.

III. A Mathematical Model of Motion Sickness

One of the main objectives of this research effort was to develop a real time processor that could read the physical parameters from a subject and compute his level of discomfort. This could then be used as a single biofeedback parameter for desensitization training. The processor has been developed.

After gathering data on several subjects (6:30) a mathematical model of motion sickness based on measured physiological parameters was required. The parameters for which the strongest correlation to subjective sickness was established were eletrosplachnogram (ESG), electronystagmogram (ENG), thoracic respiration pneumogram (Resp), galvanic skin reflex or skin resistance (GSR), and peripheral skin temperature (Temp). More detailed information is given by Drylie (4:37-55).

Data Calibration

The data recorded by the Kyowa Dengyo data recorder were digitized in the Zenith 248 using CODAS software and transferred to Asystant for analysis. The signals were sampled at 5 samples per second per channel so only low frequency components were available for the computer model. The raw data in Asystant were expressed as integers representing A/D quantizer levels multiplied by 4. First,

all data were converted to volts using

$$V = .002376(I / 4) + .011664 \quad (1)$$

where V is the voltage and I is the quantizer level stored by CODAS. The coefficients in Eq (1), as well as those in the following Eqs (2) and (3), were derived from a least squares curve fit of calibration values. After applying Eq (1), GSR was calculated by

$$\text{GSR} = 59.53(V^2) + 403.25(V) + 498.15 \quad (2)$$

and peripheral temperature was computed by

$$\text{Temp} = 0.9174(V^2) + 5.2084(V) + 83.821 \quad (3)$$

where GSR is skin resistance in kilohms and Temp is peripheral skin temperature in °F. During data digitization, the coefficients in Eqs (2) and (3) were rechecked with recorded calibration values by least squares curve fit and changed as necessary. The pallor scale was set by taking the voltage values for each channel at full exanguination and full flush. Pallor was then computed by

$$\text{PP} = (V_1 - V_0)V - (V_1 - V_0)V_0 \quad (4)$$

where PP is pallor as a fraction of full flush and V_0 and V_1 are the voltage at full exanguination and full flush respectively.

Data Preprocessing

With the input parameters converted to the proper units, the preprocessing could be done. GSR was processed by

$$GSR_P = 1 - GSR/GSR_B \quad (5)$$

The B subscript denotes the value during a control period. Temperature is similarly processed as

$$Temp_P = |Temp - Temp_B| \quad (6)$$

and pallor becomes

$$PP_P = (PP_B - PP) \quad (7)$$

The most complex preprocessing was performed on the ENG, ESG, and Resp. ENG and ESG were normalized to amplifier gains of 1500 and 600 respectively--that is, if the amplifier gains were different than those values, the output voltage was multiplied by a factor to compensate. These signals were then converted to RMS signals with the averaging done over the previous 20 seconds. The choice of 20 seconds was somewhat arbitrary, although shorter averaging times tended to give noisier signals, and longer times tended to delay the response excessively.

Asystant simulated a running 20 second RMS calculation by first squaring the entire signal, then convolving with an array of 100 points (20 seconds at 5 samples per second), each one equal to 0.01, and then taking the square root of

the resulting signal. An example of an ESG signal and the RMS result are shown in Figures 1 and 2 respectively. Each sample point in the final result is the RMS voltage taken over the last 20 seconds. The thoracic respiration signal was further processed by

$$\text{Resp}_P = \text{RMS Resp} / \text{RMS Resp}_B \quad (8)$$

In all cases, the control period is the asymptomatic period just prior to the start of head motions, and the variables subscripted with P are the processed parameters ready to fit to reports.

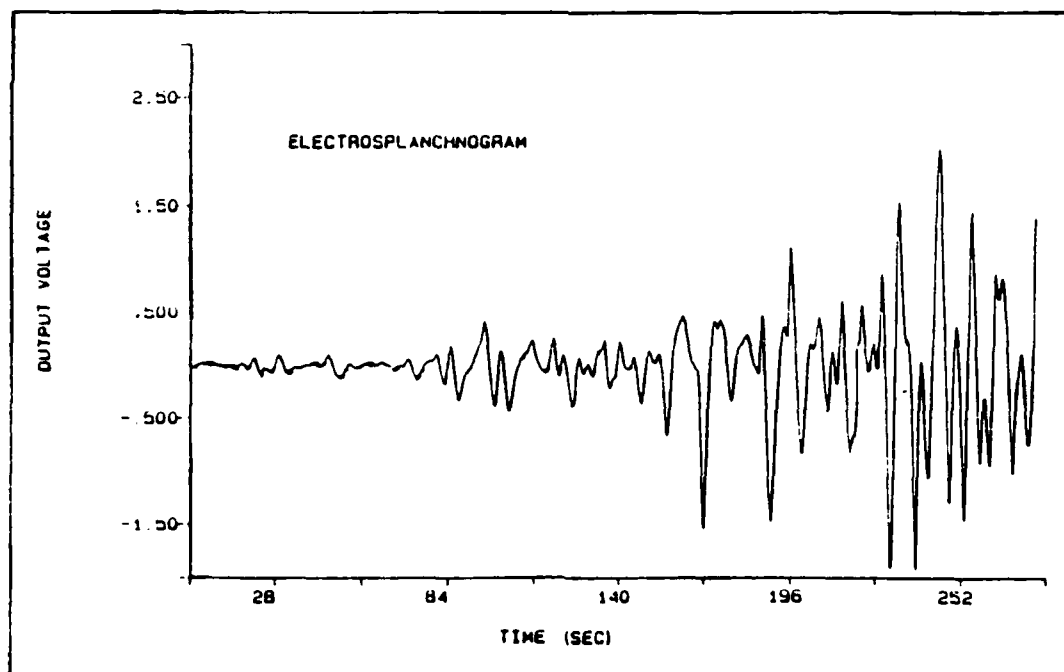


Figure 1. Electrosplachnogram Signal

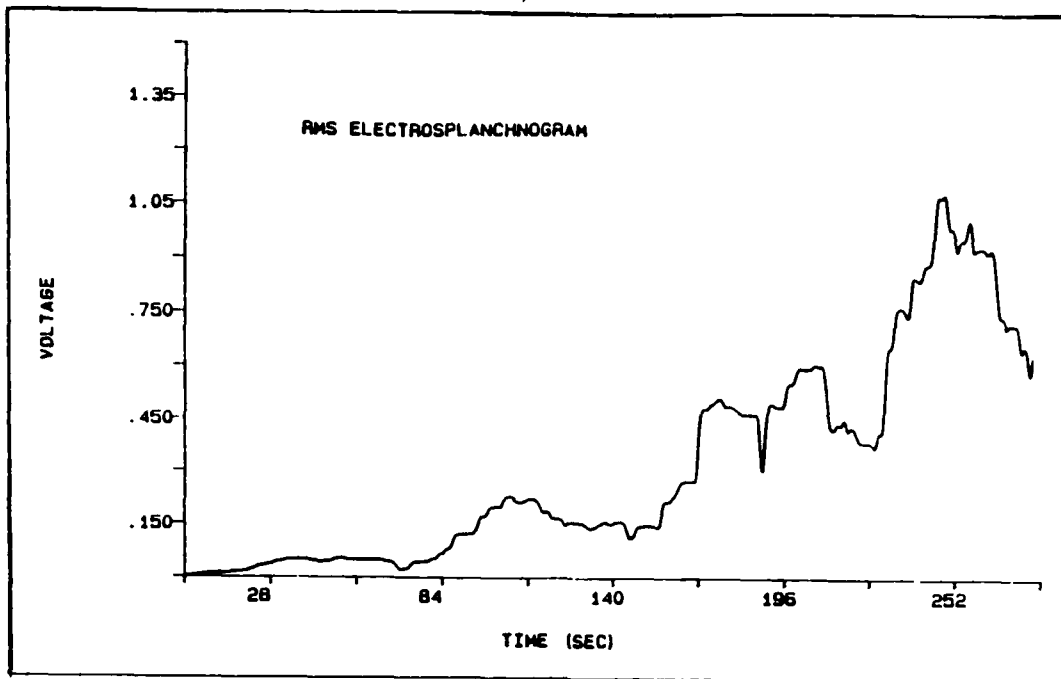


Figure 2. RMS Electrosplanchnogram Signal

The array of the subject's reports was generated by fixing its value to the number the subject reported at the time of the report and interpolating it linearly between reports. This array represented the desired output, and was the "y" array or dependent variable to which each processed parameter was fit.

Curve Fitting

Asystant. (1:Sec R9,6) The curve fitting was done by Asystant's polynomial least squares fit function. The outputs of this function are the polynomial (in this case quadratic) coefficients a, b, and c for an equation of the

form $y=ax^2+bx+c$ and a figure of merit called R^2 which is

$$R^2 = 1 - \text{Err}^2 / \text{Res}^2 \quad (9)$$

where

$$\text{Err}^2 = \sum_{i=1}^L |y_i - (ax_i^2 + bx_i + c)|^2 \quad (10)$$

$$\text{Res}^2 = \sum_{i=1}^L |y_i - 1/L \sum_{j=1}^L y_j|^2 \quad (11)$$

L is the number of points in each array (sampled parameter and reports). In most cases, every fifth sample of each array was used in the curve fit function because of memory limitations. R^2 is 0 for a data set that is completely unrelated to the fitted equation, and 1 for a perfect curve fit. The coefficients a , b , and c are the coefficients computed by the least squares algorithm. In all cases, the x_i value is the value of the preprocessed parameter sample and y_i is the value of the report array.

Average Equations. Once a single physiological parameter had been fitted for several subjects, it was necessary to combine the resulting equations into a single average equation. By averaging the corresponding coefficients from the individual equations (that is, average the a 's, the b 's and the c 's from the form ax^2+bx+c), an average equation can be obtained. This is equivalent to putting the data from all the subjects together and running the least squares fit

algorithm. However, some of the individual equations are better fits to the underlying data than others, and so are more reliable. R^2 (Eq 9) is a measure of this reliability. therefore, the average coefficients should actually be weighted by the R^2 values.

A weighted average equation was calculated by first multiplying each coefficient of each subject's fitted equation by that subject's R^2 value and dividing by the average R^2 :

$$\bar{a} = 1/M \sum_{i=1}^M (a_i R_i^2 / \overline{R^2}) \quad (12)$$

where a is the quadratic coefficient (a , b , or c), and M is the number of subjects to be averaged. Repeating this process for each parameter produced a series of averaged equations and average R^2 terms; each one describes the relationship between its particular parameter and the subjects' reports of discomfort. A composite equation was then formed by linear combination of the average equations.

The Composite Equation

To obtain the linear combination coefficients, it was first necessary to describe how tightly grouped all the individual subject's fitted curves were for each parameter. An equation averaged from a tightly grouped curve family was considered more reliable, and thus weighted more heavily in the composite equation than one from a more spread out group. The measure used was a modified standard deviation.

Curve Spread. First, the input parameter value that yielded a report value of 10 from the average equation was found. That input value was then entered into each individual subject's fitted equation, and the value

$$\text{StDev} = 1/N \sum_{i=1}^N |10 - y_i| \quad (13)$$

was obtained where N is the number of equations that went into the average equation. The StDev value was normalized by dividing

$$\text{StDev}_N = 10 / \text{StDev} \quad (14)$$

Note: if the averaged equation does not reach 10, substitute its maximum for the number 10 in Eqs (13) and (14).

Figure of Merit. Finally, a figure of merit was obtained by multiplying

$$\text{FOM} = (\text{StDev}_N) (\overline{R^2}) \quad (15)$$

Again, Eqs (13) to (15) are repeated for the averaged equation for each parameter. The figure of merit, FOM, is a number that is larger for average equations from more tightly grouped individual equations and for equations that have better average fits ($\overline{R^2}$).

The FOMs ranged from 0.8653 to 4.9116 except the facial plethysmogram FOM, with only 2 examples of correctly calibrated values, gave an FOM of 136.1. Therefore, no

plethysmograph factor was used in the final composite equation. Probably, more example, of plethysmogram data would bring its FOM more in line with the other parameters.

Linear Coefficients. With the figure of merit for each average equation, the linear combination coefficients could be calculated. First, a relative FOM was obtained:

$$FOM_R = FOM / \overline{FOM} \quad (16)$$

where $\overline{FOM}$ is the average of FOMs for all the parameters to be used in the final composite equation. Finally, the coefficients are calculated by dividing

$$F = FOM_R / K \quad (17)$$

where K is the number of individual parameters used in the composite equation. The results of Eqs (13), (15), and (17) are given in Table 1.

TABLE 1 Intermediate Linear Coefficient Results

Param	Avg R^2	StDev	FOM	F
ESG (S)	0.5605	3.1601	1.7737	0.1569
ENG (N)	0.6618	3.9539	1.5363	0.1359
Resp (R)	0.5451	2.4550	2.2208	0.1964
GSR (G)	0.8552	1.7412	4.9116	0.4344
Temp (T)	0.6757	7.8088	0.8653	0.0765

The final equation is

$$\begin{aligned}
 \text{msick} = & F_S(-1.446(\text{RMSESG}^2) + 12.97(\text{RMSESG}) + 1.141) \\
 & + F_N(-.7414(\text{RMSENG}^2) + 5.046(\text{RMSENG}) + .592) \\
 & + F_R(.1848(\text{Resp}_P^2) + 3.4587(\text{Resp}_P) - 1.537) \\
 & + F_G(7.9194(\text{GSR}_P^2) + 4.8693(\text{GSR}_P) + 1.0488) \\
 & + F_T(-1.4104(\text{Temp}_P^2) + 13.3469(\text{Temp}_P) + .6244) \quad (18)
 \end{aligned}$$

Figure 3 shows the output of this equation compared to the reported symptoms for one subject, and figure 4 shows the error signal. Similar error signals for three other subjects are given in Appendix A.

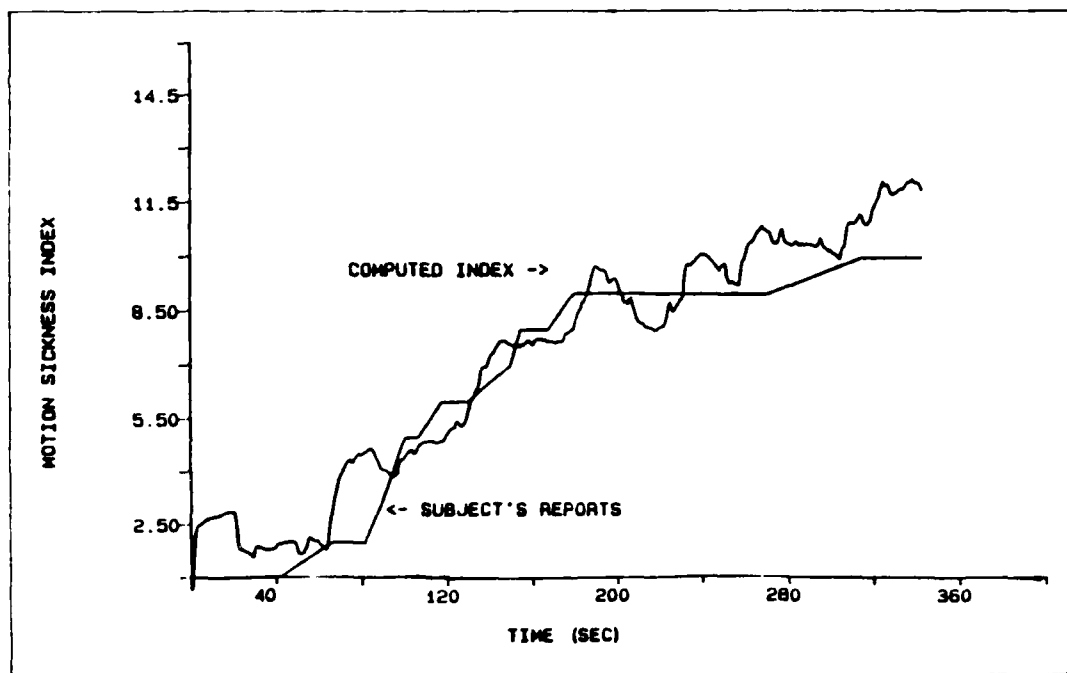


Figure 3. Computed and Reported Indices

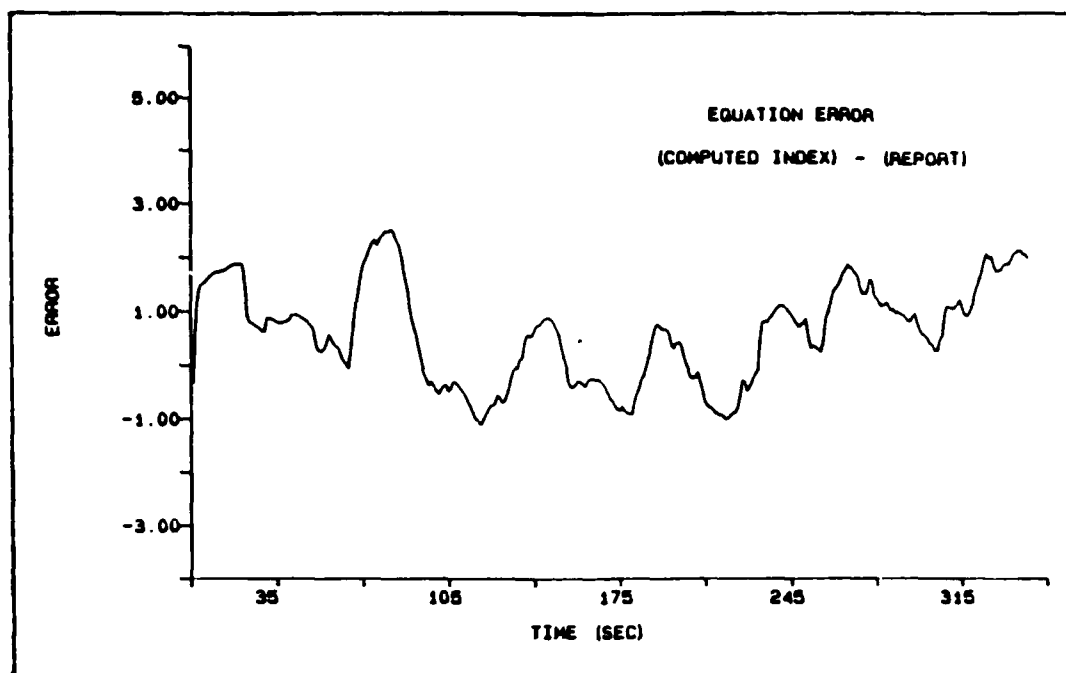


Figure 4. Equation Error

This model degrades gracefully by readjusting the linear combination coefficients in the equation to compensate for missing signals and calculating the malaise index with the signals that remain. Since each individual quadratic equation was fitted to motion sickness independently, the sum of the coefficients must equal 1. That is, in Eq (18),

$$F_S + F_N + F_R + F_G + F_T = 1 \quad (19)$$

If one of the input signals is missing, the corresponding F term is set to 0 and the others are divided by $1 - F$ so that the sum of the remaining linear combination coefficients again

equals 1:

$$F'_r = F_r / (1 - F_a) \quad (20)$$

F_a is a coefficient corresponding to an absent signal and F_r corresponds to each of the remaining signals. The computation is necessarily poorer with fewer inputs. Table 2 shows the equation's average absolute error performance with the indicated parameter missing for one subject. The error with all signals present was 0.9242.

Table 2 Equation Degraded Performance

GSR	ESG	ENG	TEMP	RESP
1.5145	1.2541	1.0077	1.1322	1.0881

IV. Real Time Implementation of the Model

This chapter will describe a real time implementation of the motion sickness model developed in the last chapter. A personal computer equipped with an 8 channel A/D converter reads the physiological signals from an experimental test subject, samples them each second, and computes and displays the subject's degree of discomfort with good accuracy. It would be a small step to send the computed output of the program through a digital to analog converter (which is also resident on the A/D board) to a display device for biofeedback training.

Specialized Equipment and Software

The computer used was a Zenith 248 (IBM PC/AT compatible) computer with an 80287 numeric data processor and a Metrabyte Dash 16 8 channel A/D converter installed. The software to drive the converter was from the Metrabyte Drivers Package written by Systems Guild of Cambridge, MA and distributed by Metrabyte. The program was written in Computer Innovations C86 C language. The functions that contain "D16()" in the name are from the Drivers package. Others that start with "key_" or "crt_" are Zenith key and screen functions and may not be portable to other IBM PC/AT compatible systems.

Adaptation of the Model

Source code of the program is in the files msick.c, calc.c, sample.c, and compute.c found in Appendix C. Through the following discussion, empty parentheses "()" indicate a function name in the C source code and braces "[]" indicate an array variable name. The electrospplanchnogram signal used came from the lower right abdomen and was presumably influenced more strongly by intestinal than by stomach signals. It was thus called "electrointestinogram" (EIG) during the experiments to distinguish it from other ESG signals. Through the remainder of this document, ESG and EIG are used interchangeably.

The program starts with function calls to initialize the A/D converters, the program's parameters and the display. Then it displays a menu allowing the operator to choose between setting up the channel assignments, calibrating the inputs, gathering baseline data, or displaying the result of the calculation.

The Model. Eq (18) from the last chapter is rewritten here:

$$\begin{aligned} \text{msick} = & F_S(-1.446(\text{RMSEIG}^2) + 12.97(\text{RMSEIG}) + 1.141) \\ & + F_N(-.7414(\text{RMSENG}^2) + 5.046(\text{RMSENG}) + .592) \\ & + F_R(.1848(\text{RESP}^2) + 3.4587(\text{RESP}) - 1.537) \\ & + F_G(7.9194(\text{GSR}^2) + 4.8693(\text{GSR}) + 1.0488) \\ & + F_T(-1.4104(\text{TEMP}^2) + 13.3469(\text{TEMP}) + .6244) \quad (21) \end{aligned}$$

In Eq (21)

$$\text{RMSEIG} = \left[\frac{1}{20} \sum_{i=1}^{20} (\text{EIG}_i)^2 \right]^{1/2} \quad (22)$$

EIG = sampled output voltage of
electrosplanchnograph amplifier

$$\text{RMSENG} = \left[\frac{1}{20} \sum_{i=1}^{20} (\text{ENG}_i)^2 \right]^{1/2} \quad (23)$$

ENG = sampled output voltage of vertical
electronystagmograph amplifier

$$\text{RESP} = \text{RMSTHO} / \text{RMSTHO}_B \quad (24)$$

$$\text{RMSTHO} = \left[\frac{1}{20} \sum_{i=1}^{20} (\text{THO}_i)^2 \right]^{1/2} \quad (25)$$

THO = voltage output of thoracic respiration
amplifier,

RMSTHO_B = RMSTHO in the control period.

$$\text{GSR} = (1 - \text{RESIST} / \text{RESIST}_B) \quad (26)$$

RESIST = skin resistance in kilohms

RESIST_B = RESIST during the control period.

$$\text{TEMP} = |T - T_B| \quad (27)$$

T = peripheral (finger) temperature,

T_B = T in the control period.

The linear coefficients F were given in table 1.

Eqs (22) through (27) represent the preprocessing and normalization operations on the physiological data before it is applied to Eq (21). One of the main constraints for picking the parameters to use was that the calculations had to be realizable in real time.

Sampling Scheme. The A/D converters sample all eight channels every 0.04 seconds and store the five latest samples in the integer array buf[]. This works in the background while the computer is doing all the other calculations. Once each second, the function sample() averages the five samples in each channel, changes that average to volts, and stores the result in the array data[]. The variables eng, eig, temp, gsr, thorsp, abdrsp, facpp, and fngpp are the indices of data[] that point to the input channel numbers for the raw data. These are assigned by the function setup().

Channel Assignment. The setup() function assigns channel numbers to the various parameters. It then calls factors() to readjust the linear combination coefficients (F). Factors() adds the coefficients of parameters that are absent, subtracts the total from 1, and divides each remaining coefficient by the difference. From Eq (20)

$$F'_{Xr} = F_{Xr} / (1 - F_{Xa}) \quad (28)$$

where F_{Xr} is the coefficient of a remaining channel and F_{Xa} represents the sum of the coefficients of absent channels.

Calibration. When the operator chooses "Calibrate", the program presents a menu of parameters to calibrate. TEMP accepts values of temperature from the operator while GSR has the standard resistance calibration values in memory. Both read the voltage at the A/D converter (or

accept input from the keyboard), and store those values in arrays. The function quadfit() computes a least squares fit to a quadratic equation and gives back the coefficients. ENG and EIG are the only two parameters of the equation that depend on the actual voltage measured at the skin, and therefore on amplifier gain. The gain calibration factors are set here. The plethysmograph voltage values for 100% flush and 0% flush (full pallor) go to linfit() and coefficients for a straight line come back. These coefficients are used to calculate actual parameter values during run(). The program stores channel assignment, calibration, and baseline data upon completing the setup, calibration, and baseline sections to facilitate rapid recovery in the event of a system crash.

Control Data. The operator sets the baseline values of the various parameters by choosing the "Control" option of the menu. The program displays the different parameters that will need baseline values and the current values of those those parameters. When the operator is satisfied the parameters show good baseline values, he selects them for use.

Preprocessing Data. The amplifier gains are applied to the nystagmogram and splanchnogram values. The equation assumes gains of 1500 and 600 respectively. Engain and eigain are the ratios of the assumed gain to the actual gain. The RMS calculation is implemented in the function rmsdat(). It is a running calculation over the samples

taken in the last 20 seconds (defined by the symbol MAXRMS). The voltage value is squared and stored in the proper place in the static arrays rabd[], rtho[], reng[], and reig[]. The results (Eqs (22), (23), and (24)) are put back in data[].

Function run() does further preprocessing. The quadval() function converts GSR and TEMP voltages to kilohms and degrees Fahrenheit respectively. The quadratic coefficients used by the function come from the calibration section of the program. Run() then applies Eqs (26) and (27) and the values are stored back in data[]. Run() also applies Eq (24) to the thoracic respiration RMS value and stores that result in data[]. The values in data[] can now be applied to Eq (21).

Output. The output display (Figure 5) is a scale numbered 1 to 10 with an arrow pointing to the latest computed value.

One remaining problem: since this equation is an average of several subjects, it may not fit an individual subject. This problem is partly alleviated by using several parameters in the equation, but not completely. Therefore, there is a provision to multiply the output by a constant factor returned by the function fudge(). If the program consistently displays a number that is greatly different from the subject's reports, the operator types the number the subject is reporting. Fudge() then computes a factor

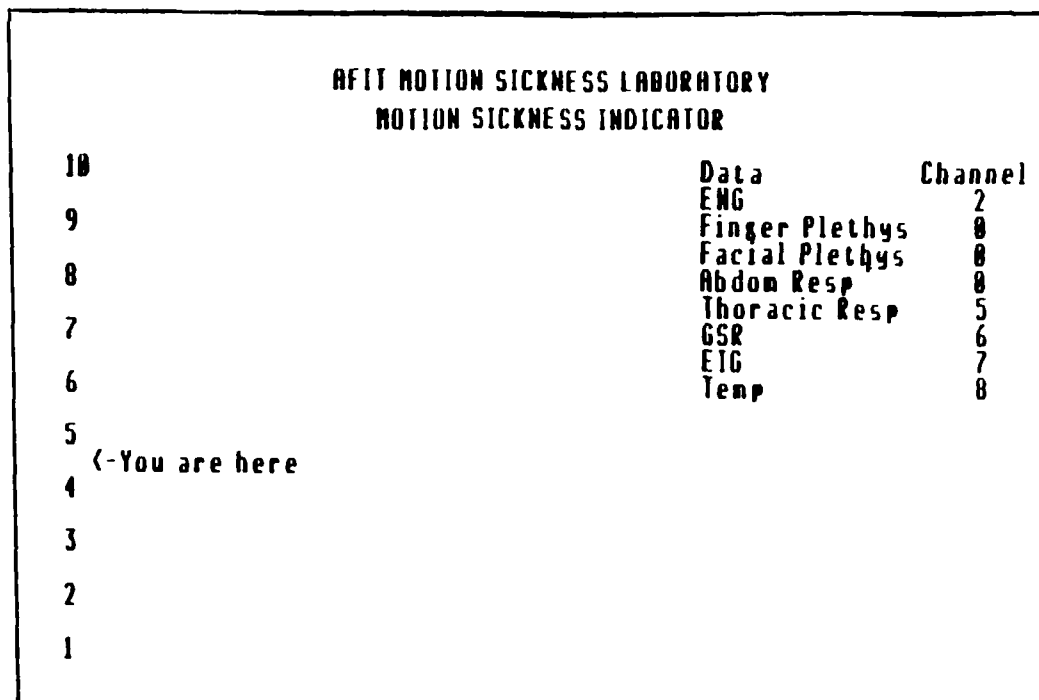


Figure 5. Output Screen Display

that will move the indicator half the distance from the computed value toward the reported value. Detailed operating instructions are in Appendix B.

The remaining inaccuracies may be dealt with in two ways. More factors could be used. Now that properly calibrated pallor data is becoming available, the pallor measurement may be added to the calculation. Also, new measurements, such as real time heart rate information may be useful. Second, it is possible that coherent patterns can be extracted from the data by other methods. Neural networks can be trained to recognize patterns, and are the subject of the next chapter.

V. Neural Network Simulation

There is enough variability between people that the equation and program described in the previous chapters do not work equally well for all subjects. However, there do seem to be recognizable patterns in the physiological data. This observation suggested using a pattern recognition approach to solve the problem. Neural nets "attempt to achieve good performance via dense interconnection of simple computational elements. In this respect, artificial neural net structure is based on our present understanding of biological nervous systems" (13:4). A type of neural net known as a multi-layer perceptron seemed to be a logical approach.

Single Node Perceptrons (13:13-14)

A single computational element or neuromime is shown in Figure 6. The output value is given by

$$y = f\left(\sum_{i=1}^N w_i x_i - \theta\right) \quad (29)$$

where

$$f(a) = \frac{1}{1 + e^{-a}} \quad (30)$$

and x represents an input vector element, w represents the connection weight, and θ is a small random threshold. N is

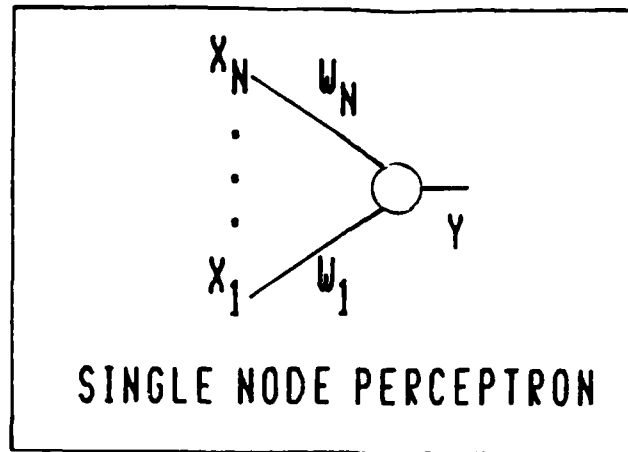


Figure 6. Single Neuromime Element

the number of elements in the input vector. It can be shown that Eq (29) describes a hyperplane boundary (a straight line if there are two inputs) in N -dimensional space between two regions. If vectors $\mathbf{x} = \{x_1, \dots, x_N\}$ which are separable into two regions are applied to the inputs, the weights can be adapted so that the hyperplane does divide the two regions of points. The training algorithm is

$$w_i(t+1) = w_i(t) + \eta [d(t) - y(t)] x_i(t) \quad (31)$$

$$0 \leq i \leq N$$

$$0 \leq \eta \leq 1$$

where d is the desired output (0 or 1). After a number of training trials, the perceptron may converge to a solution.

Multi-Layer Perceptrons (13:15-17)

It can be shown that an arrangement of several nodes in each of three layers, where all nodes in one layer (or all

inputs) are connected to all nodes of the next layer, can separate an arbitrary number of classes and regions with arbitrarily complex boundaries. This arrangement is schematically shown in Figure 7. The complexity a given arrangement can handle depends on the number of nodes in each layer.

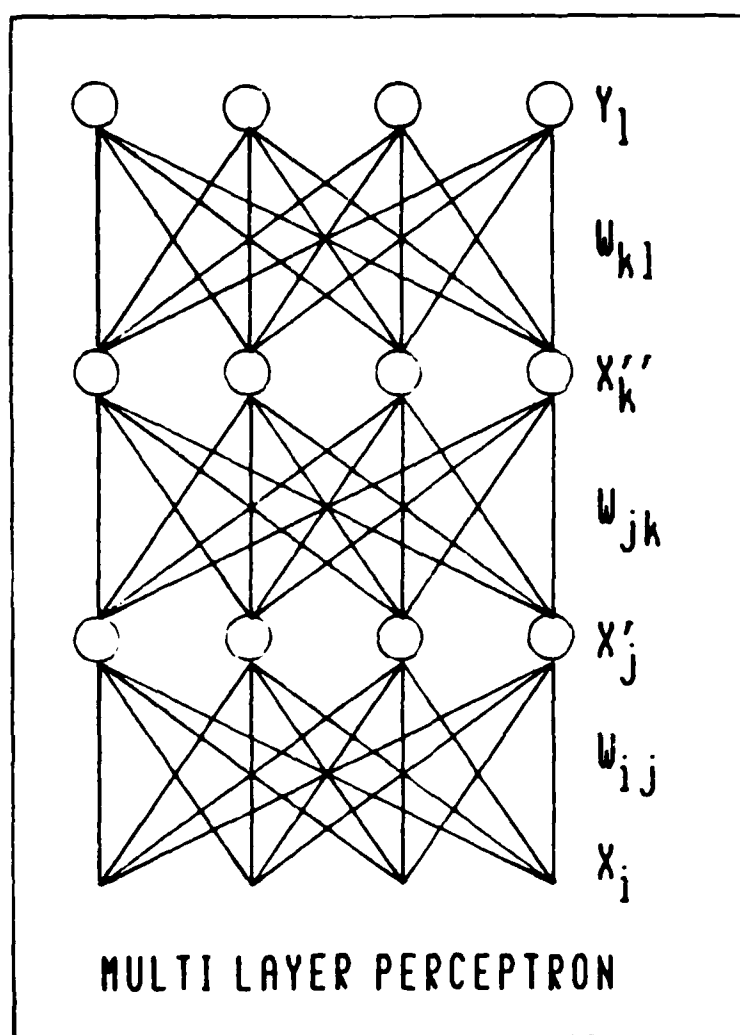


Figure 7. Neural Network

The extended training algorithm is called back propagation:

$$w_{bc}(t+1) = w_{bc}(t) + \eta \delta_c x_b \quad (32)$$

where

$$\delta_c = y_c(1 - y_c)(d_c - y_c) \quad (33)$$

if the current layer is the output where d_c is the desired output of node c and y_c is the actual output or

$$\delta_c = x_c(1 - x_c) \sum_a \delta_a w_{ca} \quad (34)$$

if the current layer is an inner or hidden layer. In Eqs (32) through (34) the subscript c denotes the current layer, while a denotes the layer above and b denotes the layer below. The θ values in Eq (29) could also be adapted by back propagation.

Perceptron Simulation

Although perceptrons are conceptually implemented as massively parallel networks of simple processors, they can be simulated on a conventional digital computer. These simulations are very computation intensive, but if the net is small enough, it may be possible to run the simulation in near real time on PC/AT class machines. The back propagation training algorithm is the most time consuming part, but once the net is trained the weights can be transferred to a real time processor.

Construction. The net simulation tested here consisted of 5 inputs, 10 nodes in the next, or first hidden, layer, 40 nodes in the second hidden layer, and 10 outputs. The inputs were the same preprocessed physiological data used for the equation model. The outputs correspond to the sickness levels from 1 to 10 reported by the subjects. The training data set was developed by averaging each four second interval from each of four subjects, and assigning an average sickness level to each input data set. This sickness level was the desired output in the training algorithm. The resulting 318 training vectors (Appendix G) were then introduced in random order to train the net. The entire set was used many times, while the gain term (η) decreased linearly from 0.2 to 2×10^{-5} over 10,000 trials and held constant thereafter.

Algorithm Implementation. The file sicknet.c (Appendix D) is the implementation of this simulation and the training algorithm. The input vector is designated by the array `input[]`, the first hidden layer is `first[]`, the second hidden layer is `second[]`, and the output layer is `output[]`. The numbers of nodes in each layer are defined by I, J, K, and L respectively. The connection weights, initially random numbers of four significant digits between -1 and 1, are contained in the two dimensional arrays `w1[][]`, `w2[][]`, and `w3[][]`.

The net simulation is implemented in the function `net()` and the back propagation algorithm is in `backprop()`. The

desired output would set the appropriate output node equal to 1 and all other output nodes equal to 0. The function `vectorize()` generates a `d` vector meeting this condition for each training vector. `Backprop()` is invoked to train the weights, but the θ values are held constant. Experiments with single node perceptrons showed the net operation was relatively insensitive to changes in θ . However, several training trials with different θ values showed 0.05 gave the best performance.

After about 95,000 training examples (applying the training vector set and back propagation about 300 times), the net could measure the training data sickness level with an average error of 0.86. In very few cases, however, did the correct output node exceed 0.5. In most cases, the values of all the nodes were small, with a peak in the 0.15 to 0.3 range in the neighborhood of the correct response. This poor definition indicates the output regions were not well separated. This may be due to the nature of the data, or the problem may lie with the neural net construction. The computed response is extracted by averaging the position values of all the nodes that exceeded a threshold set at

$$T = 1.3(\overline{out}) \quad (35)$$

where $\overline{out}$ is the average of the values of the output nodes.

The net was then incorporated into the real time indicator program with the file `net.c` (Appendix E).

Function `initfact()` was changed to include reading the trained weights from the disk (Appendix F), and the function `compute()` now propagates the input data through the net simulation and converts the output vector back to a floating point number for display. To run the simulation in the program, `net.c` is compiled and linked to the program in place of `compute.c`.

When the net makes an error, that error tends to be rather large, but there seems to be no need to correct the output with the `fudge()` function. Figure 8 shows the error signal for the same subject as Figure 4. Similar error signals for three other subjects are given in Appendix A.

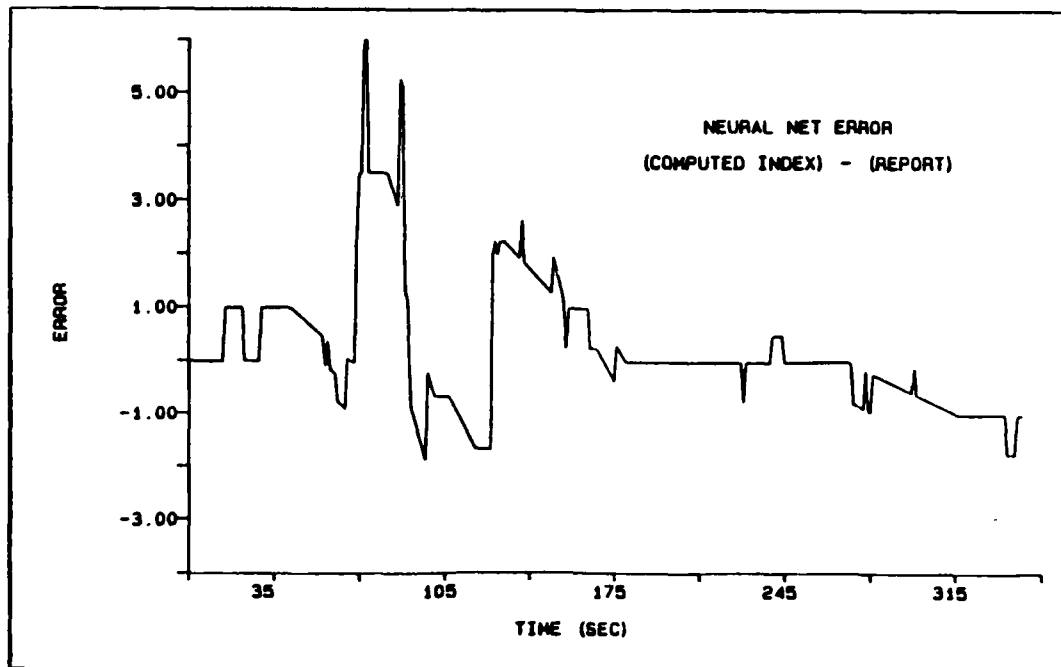


Figure 8. Net Error Signal

The program recognizes, in a sense, the general patterns but in some cases cannot place an exact value on the subject's sickness level. The number of nodes in the hidden layers determines the number of output classes (1 through 10) and the number of regions (subject types) per class the net can separate (13:16). Therefore, increasing the number of nodes in those layers might allow the net to separate the report levels better, and incidentally improve the output definition. This approach, however, would increase the training requirement, and the computation time in the indicator program.

In some cases, the net degrades gracefully. Table 3 shows the average absolute error for one subject with the indicated parameter missing. The average error when the net

Table 3 Net Degraded Performance

GSR	ESG	ENG	TEMP	RESP
1.2260	1.4600	3.3090	0.8537	1.3280

used all parameters was 0.8100. Most of the results are comparable with the equation approach, but the net degraded badly when ENG was missing. Again, it is possible that more nodes in the hidden layers will enable more graceful degradation.

In balance, the neural net approach shows promise as an alternative model in the real time processor. It is possible, however, that the net described here is not large

enough to handle the problem adequately and that a large enough simulation may not run in real time. A hardware implementation may be required.

VI. The Road Ahead

This project started with the objective of finding a reliable and repeatable way of training aircrew members to overcome motion sickness. Biofeedback was the primary method envisioned. This thesis effort focused on building a computer indicator using several physiological signals. It is possible that if the subject is presented with a nearly direct readout of his discomfort, he may be able to learn to control his symptoms more quickly and easily than with earlier methods. The only way to find out is to try.

Summary and Conclusions

Motion Sickness Equation. Equation (18) appears to correlate physiological signals to subjective malaise very well. It partially compensates for individual variability by combining five signals into a single indicator. The signals were combined by computing an average quadratic least squares fit for each, and rank ordering the different signal types according to the average quality of the curve fits and a measure of variability between subjects. This rank order was used to form a linear combination of the five fitted equations. A sixth signal, skin pallor, was not included because there were not enough examples of properly calibrated plethysmogram data for a good representation.

Real Time Implementation. The equation was adapted for implementation in a real time motion sickness indicator program running on a Zenith 248 personal computer. It appears to correlate very well with the subjective malaise reports. The output is presently displayed on the computer's screen, but could easily be routed through a digital to analog converter to drive a display in front of the subject for biofeedback training.

The indicator program degrades gracefully. The equation is adjusted to compensate for missing signals, and the program calculates the malaise index with the signals that remain. The computation is necessarily poorer with fewer inputs.

Neural Network Simulation. A neural net simulation was attempted as an alternate method of computing the malaise index. The preliminary results showed that this type of approach probably can work. When the net makes an error, it tends to be large, causing a "noisy" readout. In addition, the output layer had poor definition. These problems are probably both related to the number of nodes in the simulated net. It is possible that further experimentation will alleviate those problems.

Comparison of Neural Net and Equation. Table 4 shows the average absolute errors of both the equation and the neural net for each of four subjects. The average errors are comparable except in the case of Subject 2. There, the

Table 4 Average Absolute Errors

Subject	1	2	3	4
Equation	0.9242	1.7651	2.9500	0.7724
Net	0.8100	1.0112	2.9263	0.8956

neural net performed considerably better than the equation. This suggests that with improved architecture or training, the net may be able to handle individual differences better than the equation.

The net's degraded performance is also comparable to the equation except when the ENG signal is missing (Table 5). It seems likely that the net's performance could be improved by adding nodes to the hidden layers and employing appropriate training.

Table 5 Subject 1 Degraded Performance
Average Absolute Error

Missing	Equation	Net
None	0.9242	0.8100
GSR	1.5145	1.2260
ESG	1.2541	1.4600
ENG	1.0077	3.3090
TEMP	1.1322	0.8537
RESP	1.0881	1.3280

Recommendations

Program. The indicator program needs to be validated with more subjects. More calibrated plethysmograph data

should be gathered so that a pallor equation can be added to the composite equation. Finally, a biofeedback training program should be implemented to test whether this approach has any value.

Neural Net. The neural net model should be tested with more nodes in the hidden layers to get better definition in the output layer and better fidelity of the final net output. Other net architectures should be explored.

Miscellaneous. This thesis is not complete in itself. The reader must refer to Drylie (4) and Gaudreault (6) to completely understand the experiment and all the results. In the future, team research efforts should be written in one thesis. Artificially separating the thesis into several documents makes it harder to read, so that important and possibly beneficial work might be ignored.

Appendix A

Equation and Neural Net Error Signals

The next four pages compare the error signals from the equation and the neural net for each of four subjects. In each case, the signal is $S_e = S_o - R$ where S_e is the error signal, S_o is the output signal from the equation or neural net and R is the signal generated from the subjects' linearized reports.

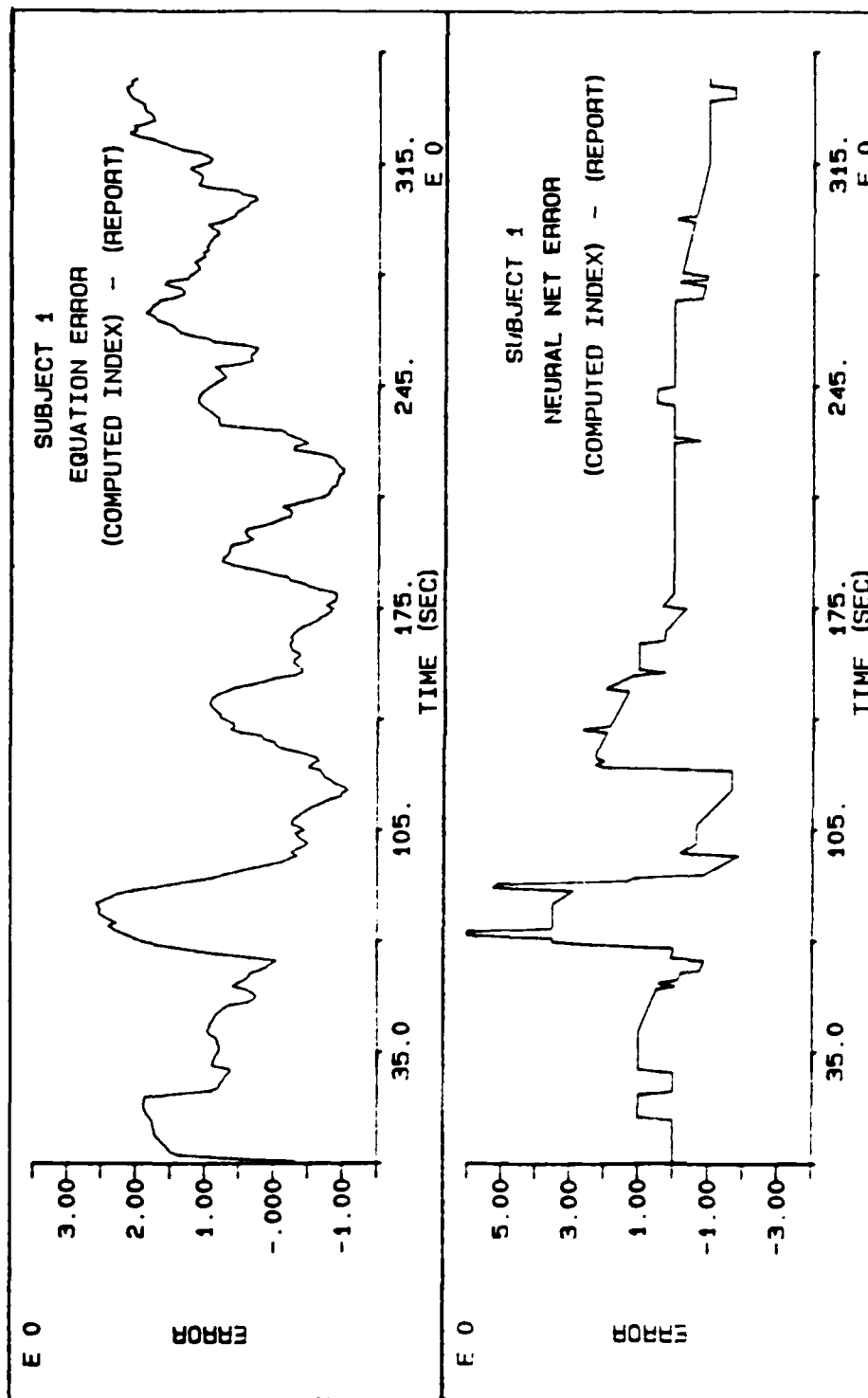


FIGURE 9. ERROR SIGNALS SUBJECT 1

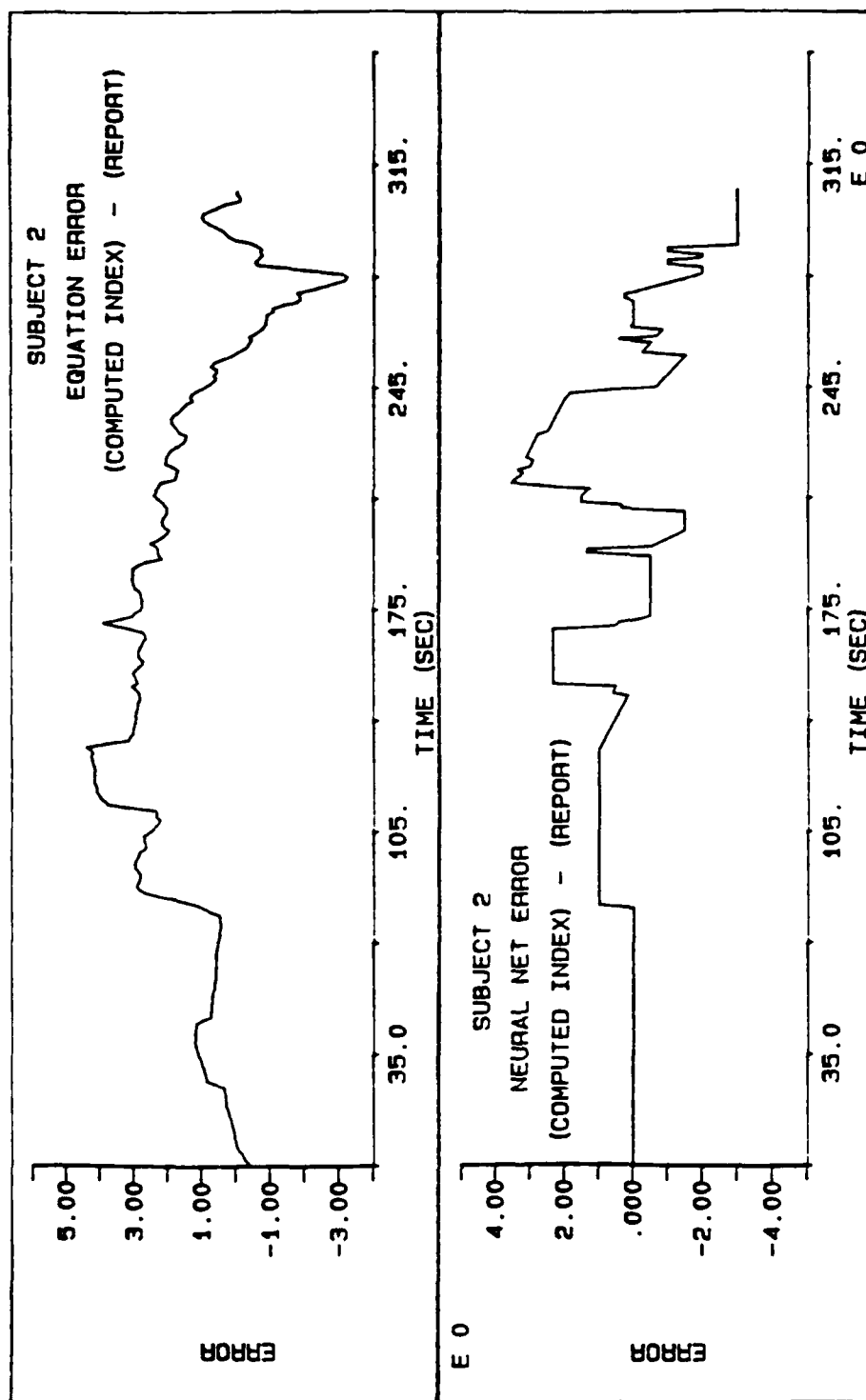


FIGURE 10. ERROR SIGNALS SUBJECT 2

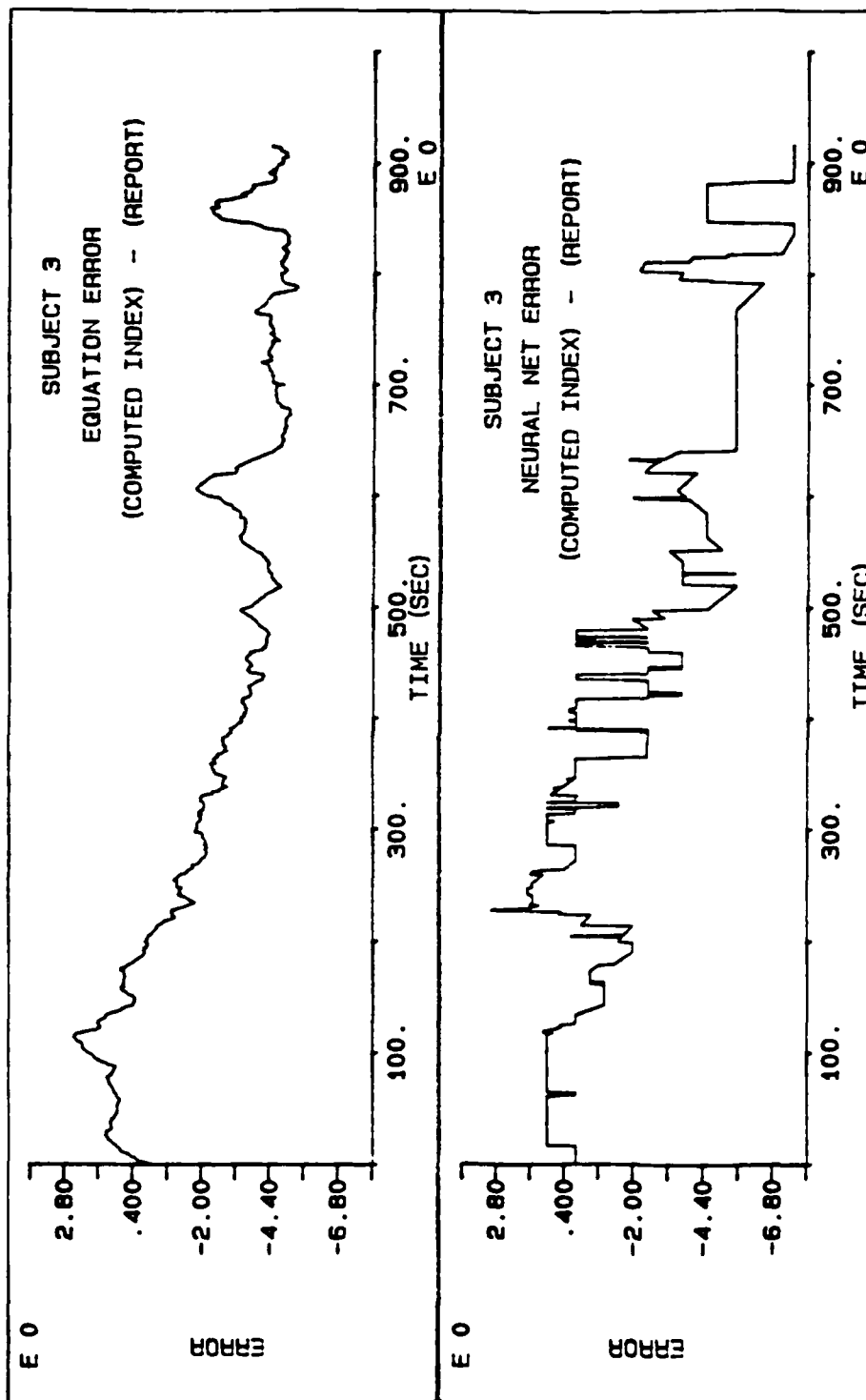


FIGURE 11. ERROR SIGNALS SUBJECT 3

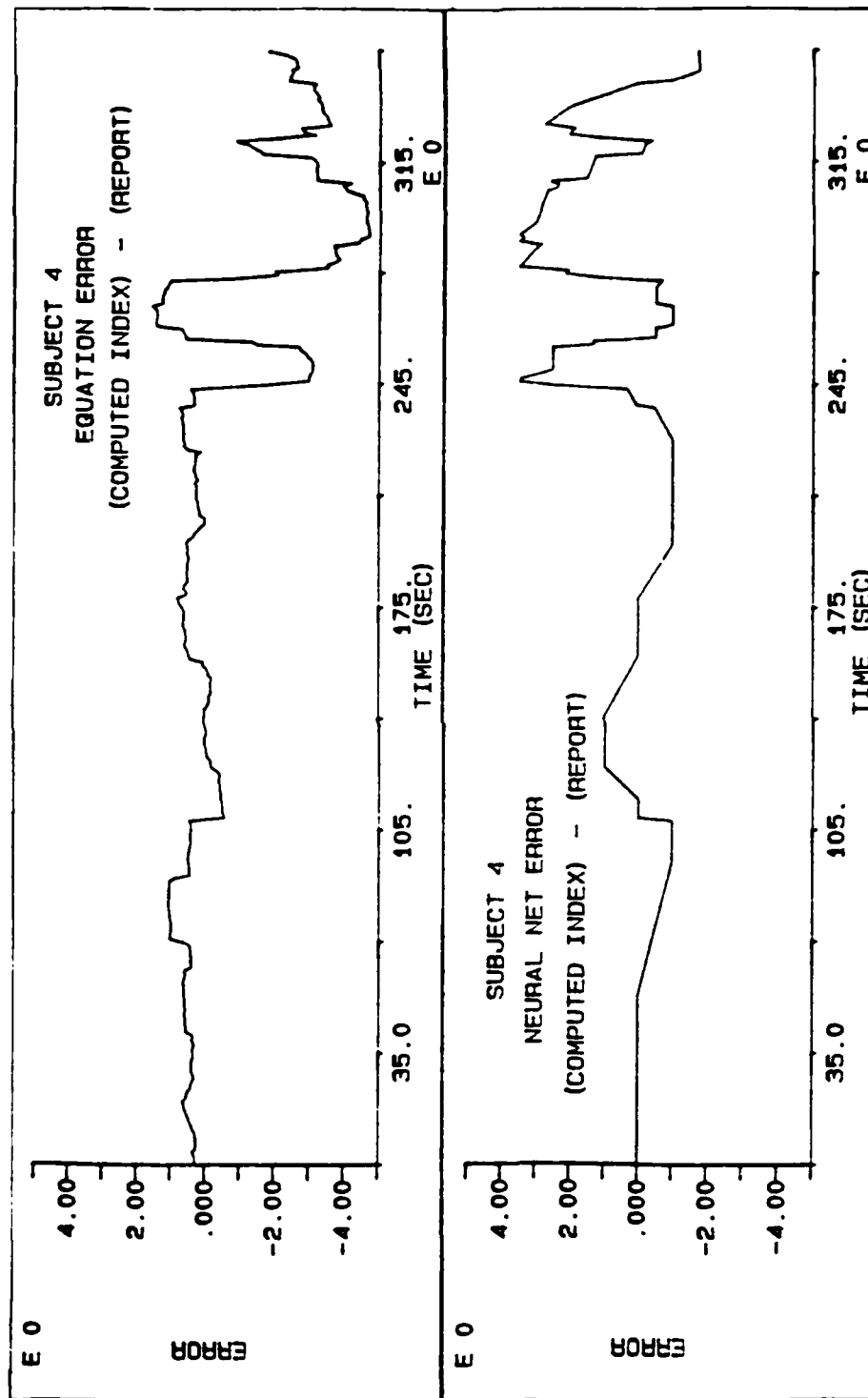


FIGURE 12. ERROR SIGNALS SUBJECT 4

Appendix B

MOTION SICKNESS INDICATOR PROGRAM Operating Instructions

Preparation. Ensure the computer is connected to the amplifiers and sensors. Channel 1 of the A/D converters must be connected to the equipment since it provides the ground for all the signals.

The program is named "Msick" and the executable file is "MSICK.EXE." In addition to this file, the same directory must also contain "MSICK.DAT". If you are using the neural net version of the program, "WEIGHTS" must also be present. MSICK.DAT contains calibration, baseline, and channel assignment data. WEIGHTS contains connection weights for the neural net simulation.

To start the program, type "msick [RETURN]", where [RETURN] means press the carriage return key. The program will begin by reading the "MSICK.DAT" file (and "WEIGHTS" if appropriate.) Then, it displays the main menu. To choose a menu option, type its number only. Pressing [ESC] in any menu or function will return you to this menu.

Setup. This option allows you to assign channel numbers to the data. The cursor will be under the number following "ENG". Type the channel number that is connected to the indicated parameter, or "0" if it is not connected or used. Press [RETURN] to leave it unchanged. Repeat for all the parameters.

Calibrate. This option gives a second menu with choices of signals to calibrate. To calibrate:

1. GSR--The program prints the calibrated resistance value (0, 20, 100, 400, 800, or 1600 kilohms) and then the voltage value it reads on the GSR channel. Press [RETURN] to accept the voltage, or type a number and [RETURN] to enter your own voltage value.
2. Temp--First, it accepts the temperature from the keyboard, then reads the voltage on the proper channel, or accepts a voltage value from the keyboard. The program accepts six temperature values.
3. Plethysmograph--The program reads voltage values from both plethysmograph channels for 0% flush. If you want to enter one voltage manually, you must enter values for both channels. Then, it goes to 100% flush and waits for [RETURN] or manually entered values.
4. ENG and EIG--The equation uses actual voltage values from both of these channels for its calculation rather than ratios or volts calibrated to some other unit of measure. Therefore, the amplifier gains are important. The equation assumes gains of 1500 and 600 for ENG and EIG respectively. Enter the actual gains if they are different.

Control. This option displays the current calibrated values for GSR, temp, RMS respiration, and pallor. When these values are stabilized to steady baseline values, hit [ESC] to use those values in the calculation. The program then goes straight into "Run" without returning to the menu. The Control function will not display any values until enough time has passed (20 seconds) for the RMS values to be accurate.

Run. This option puts you into the actual experimental run, and displays the result of the computation. If the indicator is greatly and consistently different from the

subject's reports, you can bring it more in line by pressing the numeral key corresponding to the subject's current report. You should not do this lightly, as the indicator may come back on its own.

Disaster Recovery. If the computer crashed during a run for any reason, you can recover quickly by restarting the msick program and going directly to run. As the program starts up, it will recover the latest calibration data, baseline data, and channel assignment information. Once the RMS calculations catch up, you're back in business.

Appendix C

Motion Sickness Indicator Program C Source Code

```
/*FILE MSICK.C -- MAIN PROGRAM FILE FOR THE MOTION SICKNESS
INDICATOR PROGRAM*/

/*COMPILE USING NUMERIC DATA PROCESSOR AND LINK MSICK.C,
SAMPLE.C, COMPUTE.C, AND CALC.C*/

#include "stdio.h"
#include "math.h"
#include "msick.h"
#include "time.h"

#define I 5                      /* Input vector */
#define J 10                     /* First hidden layer */
#define K 40                     /* Second hidden layer */
#define L 10                     /* Outputs */
#define RET 13                   /* Carriage Return */
#define MAXRMS 20                /* Number of seconds (samples) for RMS */
#define ESC 27                   /* Escape Character */

float first[J], second[K], output[L];          /* Nodes */
float w1[I][J], w2[J][K], w3[K][L];           /* Weights */
float thetaj, thetak, thetal;                  /* Thresholds */

main(){

extern buf[];                                /* where the data appears */
int ll = 0;
int ul = 7;
int c;
float gsrbase, tempbase, thobase, abdbase, ppbase, eigain,
    engain;
float data[9];
long j, k;

    crt_cls();                               /* clear screen */
    initfact();                             /* get initial coefficient values */
    initD16(0x310,0);                       /* initialize the A/D converter */
    intv1D16(5000L);                         /* set sample timing */
    adcranD16(ll, ul);                      /* conversion range set */
    adcdmaD16(2,40,buf);                    /*start conversions */
    read(&gsrbase, &tempbase, &thobase, &abdbase, &ppbase,
        &engain, &eigain);                 /* get latest cal data */
    factors();                              /* adjust coefficients to channel */
                                           /* assignments */
```

```

screen();                                /* write the screen */
menu();                                  /* print the main menu */

while((c = key_getc() & 0x00ff) != ESC) /* until ESC*/
/* is pressed */
    switch (c) {
        case '1':                        /* assign channels */
            screen();
            setup();
            save(gsrbase, tempbase, thobase, abdbase,
                ppbase, engain, eigain);
            screen();
            menu();
            break;
        case '2':                        /* calibrate the data */
            screen();
            calibrate(data, &engain, &eigain);
            save(gsrbase, tempbase, thobase, abdbase,
                ppbase, engain, eigain);
            screen();
            menu();
            break;
        case '3':                        /* gather baseline data */
            screen();
            baseline(data, &gsrbase, &thobase, &abdbase,
                &tempbase, &ppbase, engain, eigain);
            save(gsrbase, tempbase, thobase, abdbase,
                ppbase, engain, eigain);
        case '4':                        /* get 'em sick */
            screen();
            scale();
            run(data, gsrbase, thobase, abdbase,
                tempbase, ppbase, engain, eigain);
            screen();
            menu();
            break;
    }
    abortD16();                          /* halt conversions */
    crt_cls();
}

/* FUNCTION RUN -- DOES PREPROCESSING ON DATA, THEN SENDS IT
TO "COMPUTE" AND "DISPLAY" */

run(data, gsrbase, thobase, abdbase, tempbase, ppbase,
    engain, eigain)
    float data[];
    float gsrbase, thobase, abdbase, tempbase, ppbase,
        engain, eigain;

{
    extern int eng, eig, temp, gsr, thorsp, abdrsp, facpp,
        fngpp;                          /* channel assignment data */

```

```

extern float gsra, gsrb, gsrc, tempa, tempb, tempc,
          fgppm, fgppb, fcppm, fcppb;
/*calibration*/
/* constants */

extern long time();
extern float quadval();
extern float linval();
extern float compute();
extern float fabs();
extern float fudge();
float factor = 1;
int last = 0;
long j, k;
int t;
float x;

j = time(&k);

while((key_scan() & 0x00ff) != ESC){
    if((key_scan() & 0x00ff) >= 48 && (key_scan() &
        0x00ff) <= 57){ /* if a number is pressed */
        factor = factor * fudge(last / 2.0 + 1.0,
            ((key_getc() & 0x00ff) - 48));
        /* fudge the output */
    }
    while(key_scan() != -1) t = key_getc(); /*
clear*/
/*keyboard buffer */

while(time(&k) == j) {} /* wait one second */
sample(data);
/* turn GSR and TEMP data into*/
/* resistance and degrees */
data[gsr] = quadval(data[gsr], gsra, gsrb, gsrc);
data[temp] = quadval(data[temp], tempa, tempb,
    tempc);
/* preprocess temp and GSR */
data[temp] = fabs(data[temp] - tempbase);
data[gsr] = 1.0 - data[gsr] / gsrbase;
/* turn plethys data into*/
/* pallor fraction */
data[facpp] = linval(data[facpp], fcppm, fcppb);
data[fngpp] = linval(data[fngpp], fgppm, fgppb);
data[facpp] = ppbase - data[facpp];

/* calibrate ENG and EIG */
data[eng] = engain * data[eng];
data[eig] = eigain * data[eig];
rmsdat(data); /* take RMS values */
/* preprocess Resp data */
data[thorsp] = data[thorsp] / thobase;
data[abdrsp] = data[abdrsp] / abdbase;

```

```

        data[8] = 0.0;                                /* missing data */
                                                    /* compute and display */
        last = display(compute(data) * factor, last);
        j = k;
    }
    while(key_scan() != -1)  t = key_getc();
}

/* FUNCTION CALIBRATE -- GETS CALIBRATION INFORMATION AND
STORES CALIBRATION CONSTANTS */

calibrate(data, engain, eigain)
    float data[];
    float *engain, *eigain;
{
    int c;

    screen();
    calmenu();                                /* print calibration menu */

    while((c = key_getc() & 0x00ff) != ESC)
        switch (c) {
            case '1':                            /* calibrate GSR */
                screen();
                gsrcal(data);
                screen();
                calmenu();
                break;
            case '2':                            /* calibrate pallor */
                screen();
                plethcal(data);
                screen();
                calmenu();
                break;
            case '3':                            /* calibrate temperature */
                screen();
                tempcal(data);
                screen();
                calmenu();
                break;
            case '4':                            /* calibrate amp gains */
                screen();
                gain(engain, eigain);
                screen();
                calmenu();
        }
}

/*FUNCTION GSRCAL -- GER GSR CAL VALUES FOR QUADRATIC CURVE
FIT AND SET CONSTANTS*/

gsrcal(data)
    float data[];

```

```

{
    extern float gsra, gsrb, gsrc;
    extern int gsr;
    extern float quadfit();
    float y;
    float ans[4];
    float x[6];
    int i;
    long j, timer;
    float values[6];
        values[0] = 0;
        values[1] = 20;
        values[2] = 100;
        values[3] = 400;
        values[4] = 800;
        values[5] = 1600;

    crt_srcp(2,33,0);
    printf("GSR Calibration");
    for(i=0; i<6; ++i){
        crt_srcp(i+4, 0, 0);

        /* get the information */
        printf("Resistance = %4.0f Voltage = ",
            values[i]);
        j = time(&timer);

        while(key_scan()<0){
            while(time(&timer) <= j){}
            j = timer;
            sample(data); /* read the converters or*/
            crt_srcp(i+4, 30, 0);
            printf("%5.2f ", data[gsr]);
        }

        if((key_scan() & 0x00ff) >= 48 &&
            (key_scan() & 0x00ff) <= 57 ||
            (key_scan() & 0x00ff) == 45 ||
            (key_scan() & 0x00ff) == 44){ /*keyboard*/
            crt_srcp(i+4, 30, 0);
            printf(" ");
            crt_srcp(i+4, 30, 0);
            scanf("%f", x+i);
        }
        else{
            x[i] = data[gsr];
            while(key_scan() != -1) timer = key_getc();
        }
    }

    /* compute the constants */
    y = quadfit(6, x, values, ans);
    gsra = ans[3];

```

```

        gsrb = ans[2];
        gsrc = ans[1];
    }

/*FUNCTION TEMPCAL -- CHANGES VOLTS INTO DEGREES */

tempcal(data)
    float data[];
{
    extern float tempa, tempb, tempc;
    extern int temp;
    extern float quadfit();
    float ans[4];
    float x[6], y[6];
    float z;
    int i;
    long j, timer;

    crt_srcp(2,32,0);
    printf("Temp Calibration");
    for(i=0; i<6; ++i){
        crt_srcp(i+4, 0, 0);
        printf("Temperature = ");
        scanf("%f", y+i);          /* get temperature */
        crt_srcp(i+4, 14, 0);
        printf("%4.1f Voltage = ", y[i]);
        j = time(&timer);

        while(key_scan()<0){
            while(time(&timer) <= j){}
            j = timer;
            sample(data);
            crt_srcp(i+4, 31, 0);
            printf("%5.2f", data[temp]);
        }
        if((key_scan() & 0x00ff) >= 48 &&
            (key_scan() & 0x00ff) <= 57 ||
            (key_scan() & 0x00ff) == 45 ||
            (key_scan() & 0x00ff) == 44){          /*get input*/
            crt_srcp(i+4, 31, 0);
            printf(" ");
            crt_srcp(i+4, 31, 0);
            scanf("%f", x+i);
        }
        else{
            /* or read A/D converter */
            x[i] = data[temp];
            while(key_scan() != -1) timer = key_getc();
            /*clear keyboard buffer */
        }
    }

    /* fit the temp values */
    z = quadfit(6, x, y, ans);
    tempa = ans[3];

```

```

    tempb = ans[2];
    tempc = ans[1];
}

/*FUNCTION PLETHCAL -- CALIBRATE PLETHYSMOGRAPH WITH TWO
POINT FIT TO A STRAIGHT LINE*/

plethcal(data)
    float data[];
{
    extern int facpp, fngpp;
    extern float fcppm, fcppb, fgppm, fgppb;
    long j, timer;
    int i;
    float ans[3];
    float fgx[2], fcx[2], y[2];
        y[0] = 0;                /* full pallor */
        y[1] = 1;                /* full flush */

    crt_srcp(2,27,0);
    printf("Plethysmograph Calibration");
    j = time(&timer);
    for(i=0; i<2; ++i){
        crt_srcp(i*2+4, 0, 0);
        printf("Flush = %4.0f Voltage = Facial\n",
            y[i]);
        printf("Finger");
        while(key_scan()<0){ /* no keypresses */
            while(time(&timer) <= j) {}
            j = timer;
            sample(data);
            crt_srcp(i*2+4, 24, 0);
            printf("%5.2f", data[facpp]);
            crt_srcp(i*2+5, 24, 0);
            printf("%5.2f", data[fngpp]);
        }
        if((key_scan() & 0x00ff) >= 48 &&
            (key_scan() & 0x00ff) <= 57 ||
            (key_scan() & 0x00ff) == 45 ||
            (key_scan() & 0x00ff) == 44){ /*manual input*/
            crt_srcp(i+4, 24, 0);
            printf(" ");
            crt_srcp(i+4, 24, 0);
            scanf("%f", fcx+i);
            crt_srcp(i+5, 24, 0);
            printf(" ");
            crt_srcp(i+5, 24, 0);
            scanf("%f", fgx+i);
        }
        else{ /* or read the A/D converters */
            fgx[i] = data[fngpp];
            fcx[i] = data[facpp];
            while(key_scan() != -1) timer = key_getc();
        }
    }
}

```

```

    }
}

/* fit both plethys channels */
twoptfit(fcx, y, ans);
    fcppm = ans[1];
    fcppb = ans[2];
twoptfit(fgx, y, ans);
    fgppm = ans[1];
    fgppb = ans[2];
}

/* FUNCTION GAIN -- CALIBRATES ENG AND EIG FOR AMPLIFIER
GAIN BY RETURNING A RATIO OF PRESENT GAIN TO ASSUMED GAIN*/

gain(engain, eigain)
    float *engain, *eigain;
{
    float x;

    crt_srcp(4, 17, 0);
    printf("Nominal = 600");      /* assumed gain for EIG */
    crt_srcp(4, 0, 0);
    printf("EIG Gain = ");
    scanf("%f", &x);
    *eigain = 600 / x;

    crt_srcp(5, 17, 0);
    printf("Nominal = 1500");     /* assumed gain for ENG */
    crt_srcp(5, 0, 0);
    printf("ENG Gain = ");
    scanf("%f", &x);
    *engain = 1500 / x;
}

/*FUNCTION CALMENU -- PRINTS CALIBRATION MENU CHOICES */

calmenu()
{
    crt_srcp(2,32,0);
    printf("Calibration menu");
    crt_srcp(5,10,0);
    printf("1. GSR\n");
    printf("      2. Plethys\n");
    printf("      3. Temp\n");
    printf("      4. ENG/EIG Gain\n");
    printf("Enter Calibration Choice ");
}

/*FUNCTION BASELINE -- DISPLAYS SIGNALS FOR WHICH BASELINE
DATA IS NEEDED FOR COMPUTATION FOR THE OPERATOR TO PICK
CONTROL DATA */

baseline(data, gsrbase, thobase, abdbase, tempbase, ppbase)

```

```

float data[], *gsrbase, *thobase, *abdbase, *tempbase,
      *ppbase;
{
extern int gsr, temp, abdrsp, thorsp;
extern float gsra, gsrb, gsrc, tempa, tempb, tempc;
extern float quadval();
int i;
long j, k;

crt_srcp(2,33,0);
printf("Control Data");
crt_srcp(3,33,0);
printf("ESCAPE to Use");

i = 1;
j = time(&k);
crt_srcp(5,7,0);
printf("GSR\n");
printf("      Temp\n");
printf("      Abd Resp\n");
printf("      Thor Resp\n");
printf("      Plethys\n");

while((key_scan() & 0x00ff) != ESC){
    /* until you hit ESC */
    while(key_scan() != -1) k = key_getc();
    while(time(&k) <= j){}
    j = k;
    sample(data);
    crt_srcp(23,0,0);
    printf("%2d", i++);          /* number of seconds */

                                /*calibrate inputs */
    data[gsr] = quadval(data[gsr], gsra, gsrb, gsrc);
    data[temp] = quadval(data[temp], tempa, tempb,
                          tempc);
    rmsdat(data);
    data[8] = 0.0;              /*unused channels */

    if (i > MAXRMS){            /* RMS data is valid */
                                /* print the data */
        crt_srcp(5,0,0);
        printf("%5.0f\n", data[gsr]);
        printf("%5.1f\n", data[temp]);
        printf("%5.2f\n", data[abdrsp]);
        printf("%5.2f\n", data[thorsp]);
        printf("%5.2f\n", data[facpp]);
    }
}

data[8] = 1.0;                  /* to avoid x/0 problems */
*tempbase = data[temp];
*gsrbase = data[gsr];
*abdbase = data[abdrsp];
*thobase = data[thorsp];

```

```

*ppbase = data[facpp];
if(*gsrbase <= 100){          /* because of problems with GSR */
    gsr = -1;                 /*remove GSR measurement */
    factors();
}
data[8] = 0.0;                /* unused channels */
while(key_scan() != -1) k = key_getc();
}

/*FUNCTION SCREEN -- PRINTS HEADING AND CHANNEL DATA ON
SCREEN */

screen()
{
    extern int eng, eig, facpp, fngpp, abdrsp, thorsp, gsr,
              temp;

    crt_cls();
    crt_srcp(0, 24, 0);
    printf("AFIT MOTION SICKNESS LABORATORY");
    crt_srcp(1, 27, 0);
    printf("MOTION SICKNESS INDICATOR");
    crt_srcp(4, 56, 0);
    printf("Data");
    crt_srcp(4, 71, 0);
    printf("Channel");
    crt_srcp(6, 56, 0);
    printf("ENG");
    crt_srcp(6, 74, 0);
    if(eng == 8) printf("0");
                                /* unused data is assigned to */
                                /* channel 8; prints out as channel 0 */
    else printf("%ld", eng+1);
    crt_srcp(7,56,0);
    printf("Facial Plethys");
    crt_srcp(7, 74, 0);
    if(facpp == 8) printf("0");
    else printf("%ld", facpp+1);
    crt_srcp(8, 56, 0);
    printf("Finger Plethys");
    crt_srcp(8, 74, 0);
    if(fngpp == 8) printf("0");
    else printf("%ld", fngpp+1);
    crt_srcp(9,56,0);
    printf("Abdom Resp");
    crt_srcp(9, 74, 0);
    if(abdrsp == 8) printf("0");
    else printf("%ld", abdrsp+1);
    crt_srcp(10,56,0);
    printf("Thoracic Resp");
    crt_srcp(10, 74, 0);
    if(thorsp == 8) printf("0");
    else printf("%ld", thorsp+1);
}

```

```

    crt_srcp(11,56,0);
    printf("GSR");
    crt_srcp(11, 74, 0);
    if(gsr == 8) printf("0");
    else printf("%1d", gsr+1);
    crt_srcp(12,56,0);
    printf("EIG");
    crt_srcp(12, 74, 0);
    if(eig == 8) printf("0");
    else printf("%1d", eig+1);
    crt_srcp(13, 56, 0);
    printf("Temp");
    crt_srcp(13, 74, 0);
    if(temp == 8) printf("0");
    else printf("%1d", temp+1);
}

/*FUNCTION MENU -- PRINTS OUT MAIN MENU CHOICES */

menu()
{
    crt_srcp(5,10,0);
    printf("1. Setup\n");
    printf("          2. Calibrate\n");
    printf("          3. Control\n");
    printf("          4. Run\n");
    printf("Enter Choice ");
}

/*FUNCTION DISPLAY -- CALCULATES WHERE TO PUT THE CURSOR AND
SCROLLS THAT PORTION OF THE SCREEN */

int display(x, place)
    float x;
    int place;
{
    int new;
    if (x < 1) x = 1;
    if (x > 10) x = 10;                      /*limit the range */

    new = (x - 1) * 2 + .49;                  /*round to nearest 1/2 */
    if (new != place) crt_roll(5, 23, 3, 18, (new-place));
    return new;
}

/*FUNCTION SCALE -- PRINTS 1 - 10 SCALE ON THE SCREEN */

scale()
{
    int i;

    for(i=0; i<10; ++i){
        crt_srcp(23-i*2, 0, 0);
        printf("%2d", i+1);
    }
}

```

```

    }
    crt_srcp(23,3,0);
    printf("<-You are here");
    crt_srcp(23,1,0);
}

/*FUNCTION RMSDAT -- TAKES RUNNING RMS VALUE OF SELECTED
SIGNALS*/

rmsdat(data)
    float data[];

{
    static float rabd[MAXRMS], rtho[MAXRMS], reng[MAXRMS],
                reig[MAXRMS];          /* the last (MAXRMS)*/
                                      /* values squared*/

    static int rmsplace = 0;
    extern int abdrsp, thorsp, eng, eig;
    extern float average();
    extern float sqrt();
    int i;

                                /* square the voltages */
    rabd[rmsplace] = data[abdrsp] * data[abdrsp];
    rtho[rmsplace] = data[thorsp] * data[thorsp];
    reng[rmsplace] = data[eng] * data[eng];
    reig[rmsplace] = data[eig] * data[eig];
    ++rmsplace;
    if(rmsplace == MAXRMS) rmsplace = 0;

                                /* square root of the mean of */
                                /* the squares*/
    data[abdrsp] = sqrt(average(rabd, MAXRMS));
    data[thorsp] = sqrt(average(rtho, MAXRMS));
    data[eng] = sqrt(average(reng, MAXRMS));
    data[eig] = sqrt(average(reig, MAXRMS));
}

/*FUNCTION SAVE -- SAVES CALIBRATION CONSTANTS, CHANNEL
ASSIGNMENTS, AND BASELINE VALUES ON DISK*/

save(gsrbase, tempbase, thobase, abdbase, ppbase, engain,
    eigain)
    float gsrbase, tempbase, thobase, abdbase, ppbase,
        engain, eigain;
{
    extern float gsra, gsrb, gsrc, tempa, tempb, tempc,
                fcppm, fcppb, fgppm, fgppb;
    extern float fact_eng, fact_facp, fact_fngp, fact_abd,
                fact_tho, fact_gsr, fact_eig, fact_temp;
    extern float facts[];

    extern int fprintf();
    extern int fclose();

```

```

extern FILE *fopen();
FILE *i;

i = fopen("msick.dat", "w");

if(i > 0){
    fprintf(i, "%e ", gsra);
    fprintf(i, "%e ", gsrb);
    fprintf(i, "%e ", gsrc);
    fprintf(i, "%e ", tempa);
    fprintf(i, "%e ", tempb);
    fprintf(i, "%e ", tempc);
    fprintf(i, "%e ", fcppm);
    fprintf(i, "%e ", fcppb);
    fprintf(i, "%e ", fgppm);
    fprintf(i, "%e ", fgppb);
    fprintf(i, "%e ", gsrbase);
    fprintf(i, "%e ", tempbase);
    fprintf(i, "%e ", thobase);
    fprintf(i, "%e ", abdbase);
    fprintf(i, "%e ", ppbase);
    fprintf(i, "%e ", engain);
    fprintf(i, "%e ", eigain);
    fprintf(i, "%d ", eng);
    fprintf(i, "%d ", facpp);
    fprintf(i, "%d ", fngpp);
    fprintf(i, "%d ", abdrsp);
    fprintf(i, "%d ", thorsp);
    fprintf(i, "%d ", gsr);
    fprintf(i, "%d ", eig);
    fprintf(i, "%d ", temp);
    fclose(i);
}
else printf("Error opening the file\n");

}

/*FUNCTION READ -- READS LATEST CALIBRATION, BASELINE, AND
CHANNEL DATA */

read(gsrbase, tempbase, thobase, abdbase, ppbase, engain,
    eigain)
    float *gsrbase, *tempbase, *thobase, *abdbase, *ppbase,
        *engain, *eigain;
{
    extern float gsra, gsrb, gsrc, tempa, tempb, tempc,
        fcppm, fcppb, fgppm, fgppb;
    extern int fscanf();
    extern int fclose();
    extern FILE *fopen();
    FILE *i;

    i = fopen("msick.dat", "r");

```

```

if(i > 0){
    fscanf(i, "%f", &gsra);
    fscanf(i, "%f", &gsrb);
    fscanf(i, "%f", &gsr);
    fscanf(i, "%f", &tempa);
    fscanf(i, "%f", &tempb);
    fscanf(i, "%f", &tempc);
    fscanf(i, "%f", &fcppm);
    fscanf(i, "%f", &fcppb);
    fscanf(i, "%f", &fgppm);
    fscanf(i, "%f", &fgppb);
    fscanf(i, "%f", &gsrbase);
    fscanf(i, "%f", &tempbase);
    fscanf(i, "%f", &thobase);
    fscanf(i, "%f", &abdbase);
    fscanf(i, "%f", &ppbase);
    fscanf(i, "%f", &engain);
    fscanf(i, "%f", &eigain);
    fscanf(i, "%d", &eng);
    fscanf(i, "%d", &facpp);
    fscanf(i, "%d", &fngpp);
    fscanf(i, "%d", &abdrsp);
    fscanf(i, "%d", &thorsp);
    fscanf(i, "%d", &gsr);
    fscanf(i, "%d", &eig);
    fscanf(i, "%d", &temp);
    fclose(i);
}
else printf("Error opening the file\n");

}

/*FUNCTION SETUP -- CHANNEL ASSIGNMENT*/

setup()
{
    extern int eng, facpp, thorsp, gsr, temp, eig, fngpp,
              abdrsp;
    int i = 0;
    int x;
    int y[8];

    y[0] = eng;
    y[1] = facpp;
    y[2] = fngpp;
    y[3] = abdrsp;
    y[4] = thorsp;
    y[5] = gsr;
    y[6] = eig;
    y[7] = temp;

    crt_srcp(2,33,0);
}

```

```

printf("Set Up Channels");

while(i<8){
    crt_srcp(i+6, 74, 0);
    x = _key_getc() & 0x00ff;
    if(x == RET) ++i;                                /*status quo
    else if(x == ESC) break;                          /*return to the menu*/
    else if(x >= 48 && x <= 56){
        y[i] = x - 49;                                /*number is pressed*/
        printf("%ld", y[i++] + 1);
    }
}

eng = y[0];
facpp = y[1];
fngpp = y[2];
abdrsp = y[3];
thorsp = y[4];
gsr = y[5];
eig = y[6];
temp = y[7];

factors();
}

```

/*FUNCTION FACTORS -- ADJUSTS LINEAR COMBINATION
COEFFICIENTS FOR ABSENT SIGNALS */

```

factors()
{
    extern int eng, facpp, fngpp, thorsp, abdrsp, gsr,
              temp, eig;
    extern float fact_eng, fact_facp, fact_fngp, fact_tho,
              fact_abd, fact_gsr, fact_temp, fact_eig;
    float missing = 0;
    extern float facts[];

    fact_eng = facts[0];
    fact_facp = facts[1];
    fact_fngp = facts[2];
    fact_abd = facts[3];
    fact_tho = facts[4];
    fact_gsr = facts[5];
    fact_eig = facts[6];
    fact_temp = facts[7];

    if(gsr == -1 || gsr == 8){
        missing = fact_gsr;
        fact_gsr = 0.0;
        gsr = 8;
        /*Program reads channel 8*/
        /* if data is missing. Value*/
        /* returned will be 0. */
    }
}

```

```

if(eng == -1 || eng == 8){
    missing = missing + fact_eng;
    fact_eng = 0.0;
    eng = 8;
}
if(facpp == -1 || facpp == 8){
    missing = missing + fact_facp;
    fact_facp = 0.0;
    facpp = 8;
}
if(fngpp == -1 || fngpp == 8){
    missing = missing + fact_fngp;
    fact_fngp = 0.0;
    fngpp = 8;
}
if(thorsp == -1 || thorsp == 8){
    missing = missing + fact_tho;
    fact_tho = 0.0;
    thorsp = 8;
}
if(abdrsp == -1 || abdrsp == 8){
    missing = missing + fact_abd;
    fact_abd = 0.0;
    abdrsp = 8;
}
if(temp == -1 || temp == 8){
    missing = missing + fact_temp;
    fact_temp = 0.0;
    temp = 8;
}
if(eig == -1 || eig == 8){
    missing = missing + fact_eig;
    fact_eig = 0.0;
    eig = 8;
}
if(missing > 0){
    missing = 1 - missing;
    fact_eng = fact_eng / missing;
    fact_facp = fact_facp / missing;
    fact_fngp = fact_fngp / missing;
    fact_abd = fact_abd / missing;
    fact_tho = fact_tho / missing;
    fact_gsr = fact_gsr / missing;
    fact_eig = fact_eig / missing;
    fact_temp = fact_temp / missing;
}
}

/*FUNCTION FUDGE -- RETURNS FACTOR TO BRING COMPUTED
SICKNESS CLOSER TO REPORTS*/

```

```
float fudge(display, actual)
    float display;
    int actual;
{
    float x;
    x = (display + actual) / 2.0;    /* half the distance*/
    return(x / display);
}
```

```
/*FILE SAMPLE.C IS THE CODE TO SAMPLE THE A/D CHANNELS AND
RETURN THE VOLTAGE VALUES THEY READ*/
```

```
sample(data)
float data[];

{   int i, j;
    extern float voltage();/*function to change quantizer*/
                                /* levels to volts in file */
                                /* calc.c */

    extern int buf[];           /* buffer where the A/D */
                                /* converter stores its sampled */
                                /* values by direct memory */
                                /* access */

    abortD16();                 /* stop DMA operation */
    cvtD16(40,buf);             /* change values in buf to ints */

    for(i=0; i<8; ++i){
        for(j=8; j<40; j += 8){
            printf(""); /* required for the function */
                        /* to work -- compiler bug */

            buf[i] += buf[i+j]; /*add five samples for */
                                /* each channel */
            printf("");
        }
        data[i] = voltage(buf[i] / 5); /* average and */
                                        /* change to volts*/
    }
    adcdmaD16(2,40,buf);        /* restart sampling and DMA */
}
```

```

/*FILE COMPUTE.C -- CONTAINS FUNCTIONS TO IMPLEMENT THE
MOTION SICKNESS EQUATION */

```

```

/*FUNCTION INITFACT -- PUTS DEFAULT VALUES OF LINEAR
COMBINATION COEFFICIENTS INTO FACTS[]*/

```

```

initfact()
{
    extern float facts[];

    facts[0] = 0.1359;          /*ENG coefficient*/
    facts[1] = 0.0;             /*facial plethys coefficient*/
    facts[2] = 0.0;             /*finger plethys coefficient*/
    facts[3] = 0.0;             /*abdom resp coefficient*/
    facts[4] = 0.1964;          /*thoracic resp coefficient*/
    facts[5] = 0.4334;          /*GSR coefficient*/
    facts[6] = 0.1569;          /*EIG coefficient*/
    facts[7] = 0.0765;          /*Temp coefficient*/
}

```

```

/*FUNCTION COMPUTE -- TAKES PREPROCESSED INPUT DATA AND
RETURNS THE MOTION SICKNESS INDEX*/

```

```

float compute(data)
    float data[];

{
    extern float pow();
    extern int eng, eig, temp, gsr, thorsp;
    extern float fact_eng, fact_tho, fact_temp, fact_eig,
                fact_gsr;
    float y;

    y = fact_eng * (-.7414 * pow(data[eng], 2.0) +
        5.0463 * data[eng] - .5919) +
        fact_tho * (.1848 * pow(data[thorsp], 2.0) +
        3.4587 * data[thorsp] - 1.5368) +
        fact_temp * (-1.4104 * pow(data[temp], 2.0) +
        13.3469 * data[temp] + .6244) +
        fact_eig * (-1.4458 * pow(data[eig], 2.0) +
        12.97 * data[eig] + 1.1414) +
        fact_gsr * (7.9194 * pow(data[gsr], 2.0) +
        4.8693 * data[gsr] + 1.0488);

    return (y);
}

```

```
/*FILE CALC.C -- CONTAINS VARIOUS MATHEMATICAL FUNCTIONS*/
```

```
float quadfit(n, x, y, ans)
```

```
/* quadratic curve fit function */
```

```
/* n = number of X-Y pairs */
```

```
/* x and y are input number arrays */
```

```
/* ans is the array of output constants
```

```
for the fitted curve */
```

```
/* function returns RMS error */
```

```
float x[], y[], ans[];
```

```
int n;
```

```
{ extern float sqrt();
```

```
int m;
```

```
float a1 = 0.0;
```

```
float a2 = 0.0;
```

```
float a3 = 0.0;
```

```
float a4 = 0.0;
```

```
float b0 = 0.0;
```

```
float b1 = 0.0;
```

```
float b2 = 0.0;
```

```
float s = 0.0;
```

```
float t, v, u, w, d;
```

```
for (m=0; m<n; ++m)
```

```
{
```

```
    a1 = a1 + x[m];
```

```
    a2 = a2 + x[m] * x[m];
```

```
    a3 = a3 + x[m] * x[m] * x[m];
```

```
    a4 = a4 + x[m] * x[m] * x[m] * x[m];
```

```
    b0 = b0 + y[m];
```

```
    b1 = b1 + y[m] * x[m];
```

```
    b2 = b2 + y[m] * x[m] * x[m];
```

```
}
```

```
a1 = a1 / n;
```

```
a2 = a2 / n;
```

```
a3 = a3 / n;
```

```
a4 = a4 / n;
```

```
b0 = b0 / n;
```

```
b1 = b1 / n;
```

```
b2 = b2 / n;
```

```
d = (a2 * a4 - a3 * a3) - a1 * ((a1 * a4) -  
    (a3 * a2)) + a2 * (a1 * a3 - a2 * a2);
```

```
u = b0 * (a2 * a4 - a3 * a3) + b1 *  
    (a2 * a3 - a1 * a4) + b2 * (a1 * a3 - a2 * a2);
```

```
v = b0 * (a2 * a3 - a1 * a4) + b1 * (a4 - a2 * a2) +  
    b2 * (a2 * a1 - a3);
```

```
w = b0 * (a1 * a3 - a2 * a2) + b1 * (a1 * a2 - a3) +
```

```

        b2 * (a2 - a1 * a1);

for (m=0; m<n; ++m)
{
    t = y[m] - u/d - v/d * x[m] -
        w/d * x[m] * x[m];
    s = s + t * t;
}

s = sqrt(s);

ans[1] = u / d;
ans[2] = v / d;
ans[3] = w / d;

/* Y = ans[3] * x^2 + ans[2] * x + ans[1] */
/* RMS error = s */

return (s);
}

/*FUNCTION TWOPTFIT -- TAKES TWO POINTS AND RETURNS
COEFFICIENTS FOR A STRAIGHT LINE FUNCTION*/

twoptfit (x,y,ans)
    float x[], y[], ans[];
{
    float m, b;

    m = (y[1] - y[0])/(x[1] - x[0]); /* slope */
    b = y[1] - m * x[1]; /* y intercept */

    ans[1] = m;
    ans[2] = b;
}

/*FUNCTION QUADVAL -- ACCEPTS INPUT VALUE AND QUADRATIC
COEFFICIENTS AND RETURNS OUTPUT Y VALUE.*/

float quadval(dat, a, b, c)
    float dat, a, b, c;

{
    float x;
    extern float pow();

    x = a * dat * dat + b * dat + c;

    return x;
}

/*FUNCTION LINVAL -- ACCEPTS INPUT VALUE AND STRAIGHT LINE
COEFFICIENTS; RETURNS OUTPUT Y VALUE*/

```

```
float linval(dat, m, b)
    float dat, m, b;
```

```
{    float x;

    x = m * dat + b;
    return x;
}
```

/*FUNCTION AVERAGE -- ACCEPTS INPUT VECTOR AND NUMBER OF
POINTS IN THE VECTOR. RETURNS THE AVERAGE OF THE VALUES IN
THE VECTOR.*/

```
float average(vector, number)
    int number;
    float vector[];
{    int i;
    float x = 0.0;

    for(i=0; i<number; ++i){
        x = x + vector[i];
    }

    x = x / number;
    return x;
}
```

/*FUNCTION VOLTAGE -- ACCEPTS THE QUANTIZER LEVEL FROM THE
A/D CONVERTER, RETURNS FLOATING POINT VOLTAGE*/

```
float voltage(x)
    int x;
{    float y;

    y = .002376 * x + .011664;

    return(y);
}
```

/*FILE MSICK.H -- GLOBAL VARIABLES DEFINED*/

```
int eng, facpp, fngpp, abdrsp, thorsp, eig, gsr, temp;
int buf[56];
float gsra, gsrb, gsrc, tempa, tempb, tempc, fcppm, fcppb,
      fgppm, fgppb;
float fact_eng, fact_facp, fact_fngp, fact_abd, fact_tho,
      fact_gsr, fact_eig, fact_temp;
float facts[8];
```

Appendix D

Neural Net Simulation Program C Source Code

```
/*FILE SICKNET.C -- NEURAL NETWORK SIMULATION AND TRAINING
ALGORITHM */

#include <math.h>
#include <stdio.h>
#define I 5 /* nr. of inputs */
#define J 10 /* nodes in first hidden layer */
#define K 40 /* nodes in second hidden layer */
#define L 10 /* output nodes */
#define END 50 /* number of reps with training vector set */
#define VECTORS 318 /* nr. of training vectors in the set */

float input[I], first[J], second[K], output[L];
float w1[I][J], w2[J][K], w3[K][L];
float thetaj, thetak, thetal;

main()
{
    extern float advance();
    extern FILE *fopen();
    int outvec[L];
    float x, eta;
    int i, j, k, l, ans;
    FILE *dat;

    initial(); /* get original weights */
               /* random numbers between -1 and 1 */

    for(i=0; i<END; ++i){
        dat = fopen("training.dat", "r");

        for(j=0; j<VECTORS; ++j){
            fscanf(dat, "%f", &x);
            ans = x;
            vectorize(ans, outvec, L); /*set up correct*/
                                       /* output vector */
            for(k=0; k<I; ++k){
                fscanf(dat, "%f", input+k);
            }
            eta = advance(i, j, VECTORS);
                    /*compute gain factor */

            net(); /*run the simulation*/
            backprop(outvec, eta); /*train the weights*/
        }
    }
}
```

```

        fclose(dat);
        wtsave();                                /* save the weights from */
                                                    /* this iteration */
    }
}

/*FUNCTION WTSAVE -- SAVE TRAINED WEIGHTS TO A DISK FILE*/

wtsave()
{
    extern FILE *fopen();
    FILE *dat;
    int i, j, k, l;

    dat = fopen("weights", "w");

                                                    /*save the thresholds */
    fprintf(dat, "%f %f %f\n", thetaj, thetak, thetal);

    for(i=0; i<I; ++i){
        for(j=0; j<J; ++j){
            fprintf(dat, "%f ", w1[i][j]);
        }
        fprintf(dat, "\n");
    }

    for(j=0; j<J; ++j){
        for(k=0; k<K; ++k){
            fprintf(dat, "%f ", w2[j][k]);
        }
        fprintf(dat, "\n");
    }

    for(k=0; k<K; ++k){
        for(l=0; l<L; ++l){
            fprintf(dat, "%f ", w3[k][l]);
        }
        fprintf(dat, "\n");
    }
    fclose(dat);
}

/*FUNCTION INITIAL -- READ INITIAL RANDOM WEIGHTS FROM DISK
*/

initial()
{
    extern FILE *fopen();
    FILE *dat;
    int i, j, k, l;

    dat = fopen("weights", "r");
    fscanf(dat, "%f %f %f", &thetaj, &thetak, &thetal);
    for(i=0; i<I; ++i){

```

```

        for(j=0; j<J; ++j){
            fscanf(dat, "%f ", &w1[i][j]);
        }
    }

    for(j=0; j<J; ++j){
        for(k=0; k<K; ++k){
            fscanf(dat, "%f ", &w2[j][k]);
        }
    }

    for(k=0; k<K; ++k){
        for(l=0; l<L; ++l){
            fscanf(dat, "%f ", &w3[k][l]);
        }
    }
    fclose(dat);
}

/*FUNCTION NET -- NEURAL NET SIMULATION*/

net()
{
    extern float sigmoid();
    float x;
    int i, j, k, l;

    for(j=0; j<J; ++j){
        x = 0.0;
        for(i=0; i<I; ++i){
            x = x + w1[i][j] * input[i] - thetaj;
        }
        first[j] = sigmoid(x);
    }

    for(k=0; k<K; ++k){
        x = 0.0;
        for(j=0; j<J; ++j){
            x = x + w2[j][k] * first[j] - thetak;
        }
        second[k] = sigmoid(x);
    }

    for(l=0; l<L; ++l){
        x = 0.0;
        for(k=0; k<K; ++k){
            x = x + w3[k][l] * second[k] - thetal;
        }
        output[l] = sigmoid(x);
    }
}

/*FUNCTION BACKPROP -- BACK PROPAGATION ALGORITHM TO TRAIN

```

THE NET*/

```

backprop(outvec, eta)
    float eta;
    int outvec[];
    {
        float sum1[J], sum2[K];
        float delta;
        int i, j, k, l;
        for(j=0; j<J; ++j) sum1[j]=0;
        for(k=0; k<K; ++k) sum2[k]=0;
        for(l=0; l<L; ++l){
            delta = (output[l] * (1.0 - output[l]) *
(outvec[l] - output[l]));
            for(k=0; k<K; ++k){
                sum2[k] = sum2[k] + delta * w3[k][l];
                w3[k][l] = w3[k][l] + eta * delta * second[k];
            }
        }
        for(k=0; k<K; ++k){
            delta = second[k] * (1.0 - second[k]) * sum2[k];
            for(j=0; j<J; ++j){
                sum1[j] = sum1[j] + delta * w2[j][k];
                w2[j][k] = w2[j][k] + eta * delta * first[j];
            }
        }
        for(j=0; j<10; ++j){
            delta = first[j] * (1.0 - first[j]) * sum1[j];
            for(i=0; i<I; ++i){
                w1[i][j] = w1[i][j] + eta * delta * input[i];
            }
        }
    }

```

/*FUNCTION SIGMOID -- NONLINEARITY FOR NODE OUTPUTS*/

```

float sigmoid(x)
    float x;
    {
        float y;

        y = 1 / (1 + exp(-x));
        return (y);
    }

```

/*FUNCTION ADVANCE -- INCREMENTS THE GAIN FUNCTION FOR BACK PROPAGATION*/

```

float advance(a, b, c)
    int a, b, c;
    {
        float t;

```

```

t = a * c + b;
if(t >= 10000) t= 0.2000E-004;
else t = (.2 * (1-(t / 10000)));
return (t);
}

```

```

/*FUNCTION VECTORIZE -- ACCEPTS AN INTEGER FROM 0 TO NR AND
RETURNS AN NR ELEMENT ARRAY WHERE EACH ELEMENT IS 0 EXCEPT
THE POSITION CORRESPONDING TO ANS = 1 */

```

```

vectorize(ans, outvec, nr)
int ans, nr, outvec[];
{
int i;

for(i=0; i<nr; ++i) outvec[i] = 0;
outvec[ans - 1] = 1;
}

```

Appendix E

Neural Net Real Time Implementation C Source Code

```
/*FILE NET.C -- FOR A NEURAL NET VERSION OF THE MSICK  
PROGRAM, COMPILE THIS FILE AND LINK IT IN PLACE OF  
COMPUTE.C*/
```

```
#include <stdio.h>  
#define I 5  
#define J 10  
#define K 40  
#define L 10
```

```
/*FUNCTION INITFACT -- INITIALIZE FACTS[] AND WEIGHTS*/
```

```
initfact()  
{   extern float facts[];  
  
    facts[0] = 0.1359e0;  
    facts[1] = 0.0E0;  
    facts[2] = 0.0E0;  
    facts[3] = 0.0E0;  
    facts[4] = 0.1964E0;  
    facts[5] = 0.4334E0;  
    facts[6] = 0.1569E0;  
    facts[7] = 0.0765E0;
```

```
    initial();      /* load trained weights from sicknet */
```

```
}
```

```
/*FUNCTION COMPUTE -- RETURNS MOTION SICKNESS INDEX*/
```

```
float compute(data)  
    float data[];
```

```
{   extern float calc();  
    float y;
```

```
    net(data);                      /* run net simulation */  
    y = calc(); /* change output vector to index number */
```

```
    return(y);
```

```
}
```

```
/*FUNCTION INITIAL -- READ TRAINED WEIGHTS FROM FILE */
```

```
int initial()
```

```

{   extern FILE *fopen();
    FILE *dat;
    extern float w1[][J], w2[][K], w3[][L];
    extern float thetaj, thetak, thetal;
    int i, j, k, l;

    dat = fopen("weights", "r");
    fscanf(dat, "%f %f %f", &thetaj, &thetak, &thetal);
    for(i=0; i<I; ++i){
        for(j=0; j<J; ++j){
            fscanf(dat, "%f ", &w1[i][j]);
        }
    }

    for(j=0; j<J; ++j){
        for(k=0; k<K; ++k){
            fscanf(dat, "%f ", &w2[j][k]);
        }
    }

    for(k=0; k<K; ++k){
        for(l=0; l<L; ++l){
            fscanf(dat, "%f ", &w3[k][l]);
        }
    }
    fclose(dat);
}

/*FUNCTION NET -- NEURAL NET SIMULATION*/

net(in)
    float in[];
{   extern float sigmoid();
    extern float first[J], second[K], output[L];
    extern float w1[][J], w2[][K], w3[][L];
    extern float thetaj, thetak, thetal;
    extern int gsr, eig, eng, temp, thorsp;
    float x;
    int i, j, k, l, ans;

    for(j=0; j<J; ++j){
        x = 0.0;
        x = x + w1[gsr][j] * in[gsr] - thetaj;
        x = x + w1[eig][j] * in[eig] - thetaj;
        x = x + w1[eng][j] * in[eng] - thetaj;
        x = x + w1[temp][j] * in[temp] - thetaj;
        x = x + w1[thorsp][j] * in[thorsp] - thetaj;

        first[j] = sigmoid(x);
    }
}

```

```

for(k=0; k<K; ++k){
    x = 0.0;
    for(j=0; j<J; ++j){
        x = x + w2[j][k] * first[j] - thetak;
    }
    second[k] = sigmoid(x);
}

for(l=0; l<L; ++l){
    x = 0.0;
    for(k=0; k<K; ++k){
        x = x + w3[k][l] * second[k] - thetal;
    }
    output[l] = sigmoid(x);
}

/*FUNCTION SIGMOID -- NODE NONLINEARITY*/

float sigmoid(x)
{
    float x;
    float y;
    extern float exp();

    if(x < -20) y = 0;
    else if(x > 20) y = 1;
    else y = 1 / (1 + exp(-x));
    return (y);
}

/*FUNCTION CALC -- CHANGE THE OUTPUT VECTOR TO A FLOATING
POINT NR*/

float calc()
{
    int out[L];
    int i;
    int count = 0;
    float thresh;
    float score = 0.0;
    float avg = 0.0;
    float max = 0.0;

    for(i=0; i<L; ++i){
        avg = avg + output[i];
        if(output[i] > max) max = output[i];
    }
    avg = avg / L;
    if(max >= .5) thresh = .5;
    else thresh = avg * 1.3;

    for(i=0; i<L; ++i){
        if(output[i] >= thresh){

```

```

        score = score + i + 1.0;
        count = count + 1;
                /*number of nodes above thresh */
    }
}
if(count > 0) return( score / count);
else return(0.0);      /* to avoid divide by 0 */
}

```

Appendix F

Neural Net Weights

These are the weights after the net was trained. They are formatted to be loaded by the function initial.

```
0.050000 0.050000 0.050000
-1.079754 0.397362 0.642471 0.339011 0.225112 -1.283468
-0.924953 1.103809 0.315904 -1.409071
-0.866108 0.722154 0.334262 -0.845744 -0.565197 -0.238325
0.577148 1.160294 1.955390 -0.035808
-0.982329 -0.154164 1.131467 -0.297720 -1.416551 -1.548174
-0.650107 1.247586 0.137664 -1.389917
-1.044137 0.471929 0.911274 -0.874905 0.301148 1.434527
-0.759823 -0.024612 -1.574444 -0.503947
0.405577 -0.024487 0.145775 -1.056779 -0.678607 1.211294
-0.554243 -0.114669 -0.394248 0.306093
-1.095977 0.842472 -0.487451 -0.406159 0.276573 0.419930
-1.468692 0.323960 -0.344719 0.487745 -1.193820 0.543513
0.776049 -0.092565 -1.264917 -0.553825 -0.974610 0.213631
-0.639952 0.042742 1.146738 0.351694 0.629713 0.381352
-0.072170 0.960733 -0.131944 -0.197966 0.543715 -0.842866
0.315885 0.439095 0.741966 1.387904 0.817269 0.662875
0.771700 -0.581290 -0.734963 1.009922
-0.385300 0.363077 -0.351991 0.816001 0.439380 -0.358664
-0.145953 -0.329500 -0.605346 0.491592 0.632398 0.682267
-0.554270 -0.190201 0.048035 0.790123 0.022577 -0.659563
0.222512 0.542629 -0.164548 -0.580540 -0.257500 0.512454
-0.224635 0.312813 0.865226 -0.108978 0.172081 -0.070278
0.121593 -0.389647 0.840170 -0.949835 -0.597163 0.429470
0.582216 1.012707 -0.651631 -0.531404
0.553247 -1.027112 -0.137848 0.033037 -0.025600 0.180884
1.274724 -0.209107 0.352204 -0.202829 0.955148 -0.519854
-0.729532 -0.888504 0.715326 0.492791 0.334352 -0.103879
-0.232116 1.004693 0.098685 -0.209398 0.399507 -0.884801
-0.895376 0.145573 -1.018597 -0.585808 0.425269 0.903843
0.248966 0.473034 -0.921944 -0.147577 0.679318 -0.396408
0.129673 -0.236578 0.536842 -0.510166
0.691391 0.666721 0.228530 -0.205695 0.070450 0.379701
-1.108514 -0.438902 0.715623 0.424322 -0.659434 -0.483925
-0.762641 0.838502 0.467031 0.045205 0.615207 -0.183182
-0.539064 -1.004198 0.847597 -0.657637 -0.903449 -0.100253
-0.015467 0.984093 0.747298 -0.382135 0.178577 -0.030889
0.349825 -0.292867 0.807472 0.135220 0.134684 0.438469
0.763879 0.041736 0.103190 -0.309365
-0.664404 0.924281 -0.664293 -1.099882 -0.100571 0.202650
-1.129005 0.694958 -0.564081 0.017047 -0.369515 0.103068
0.586917 0.442019 -0.569150 -0.535359 0.450225 0.207448
0.568134 -0.370116 0.991908 0.470529 0.259988 -0.431193
0.556169 -0.240785 -0.540027 0.753594 -0.047893 0.183539
```

0.899669 0.362608 0.085726 0.375024 -0.228956 -0.367306
0.015700 0.440825 0.712937 0.807908
-0.716808 -0.288183 -0.897734 0.338306 0.758128 0.575761
-0.206398 -0.318369 -0.575412 -0.316384 -0.453036 0.190611
0.472058 -1.064345 0.284718 0.366341 -0.970922 -0.360905
1.213737 0.353565 0.776366 -0.613857 -1.106530 1.283658
-0.908207 0.808711 0.597915 0.211599 0.868258 1.398865
0.482195 -0.542108 1.024276 1.349374 0.370303 0.010071
-1.100153 -0.045517 -0.301864 -0.006499
0.197451 0.294020 0.540993 -0.186396 -1.127492 0.138677
-0.704663 0.550511 -0.707086 -0.467363 -0.106324 -0.172510
0.555782 -0.371793 0.141946 0.074696 -0.904714 0.143666
-0.461148 0.454725 0.634250 -0.551452 0.964787 -0.542853
-0.644871 0.327503 1.014401 0.904415 -0.903189 -0.351975
-0.798218 0.357861 0.319784 1.267768 -0.229130 0.006668
-0.476934 -0.796813 0.257845 -0.201256
1.260002 0.013546 -0.131799 0.034441 0.042817 -0.287978
1.295206 -0.517382 0.190440 -0.354844 0.677967 -0.417523
0.611599 0.339350 0.265870 -0.021088 -0.034778 0.111552
0.728013 0.482087 -0.837513 -0.501849 0.383301 0.051082
-0.808448 -1.383680 -0.518579 -0.740165 -0.691435 -0.417751
0.138322 -0.954298 -0.090045 -1.556317 0.005478 0.602680
0.734632 -0.382220 -0.461853 0.434226
0.586882 0.939086 -0.470361 0.925420 0.379584 0.057327
0.708202 -0.316723 -0.377291 0.831753 -0.145182 0.334979
-0.534497 -0.457968 -0.489097 -0.486064 0.404792 -0.015699
-1.180356 -0.492345 -0.771540 -0.819139 -0.180052 -0.659853
-0.523380 0.923177 0.803158 0.956147 -0.986463 -1.126414
0.871728 0.557274 -0.202380 0.360841 0.180828 -0.719368
0.318182 1.466492 0.625833 -0.307420
-1.098900 1.035528 0.078347 -0.549245 -0.737646 -0.109926
-1.124732 0.657653 -0.580281 0.730918 -0.373158 0.001882
0.603442 -0.473559 -0.122005 -0.834726 -0.420908 -0.552252
-0.774911 0.539427 -0.066663 0.202814 0.288482 0.679088
-0.514950 -0.249940 0.812157 0.719124 0.110291 -0.602652
-0.095757 0.671072 0.340302 1.396695 -0.465651 -0.674928
0.829015 -0.651499 -1.270774 0.453254
-0.929061 -0.809647 -1.328627 0.843918 0.290778 0.967169
0.503987 1.022338 1.002529 -0.130198
1.300973 0.001888 -0.843523 -0.967480 -1.034742 -0.967190
-0.824644 0.495429 -0.234984 0.445708
-0.035994 0.401637 0.241201 0.825248 0.057065 0.686898
0.390529 0.506339 0.113672 0.230849
-0.994746 0.787350 0.607853 -0.520841 -0.803644 -0.217784
0.858261 -0.744401 0.506217 0.737408
-0.516399 -0.790327 0.926604 -0.012894 0.541214 0.111144
-0.662873 0.334339 -0.598533 0.280466
0.800797 0.684212 -0.169393 0.255553 0.375786 0.111144
-0.373066 0.831100 -0.055451 -0.185759
-2.697945 -0.351379 -1.007822 0.316185 0.111144
-0.366977 1.096257 1.051676 0.247754
0.925274 -0.195056 0.509215 0.711144

AO-A189 674

MOTION SICKNESS: QUANTITATIVE ALGORITHMIC MALAISE
INDICATION IN REAL TIME(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING E L FIX
UNCLASSIFIED DEC 87 AFIT/GE/ENG/87D-18 F/G 6/10

2/2

NL





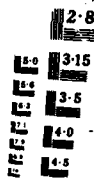
1-C



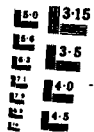
1-1



1-25



2-8



3-15



3-5



4-0



2-2



2-0



1-6



1-4



1-6

-0.222546 -0.038308 -1.189092 0.414123
 0.029509 0.584262 -0.027754 -0.005620 0.622370 0.076722
 0.000907 -0.499513 0.283075 0.236899
 0.446047 -0.037884 0.155228 -0.941311 0.710496 -1.002161
 0.571846 -0.369540 0.479258 0.076893
 -1.773423 -0.708455 -0.021843 -0.358488 -0.992986 -0.316583
 -0.599067 0.269993 0.868978 0.360468
 0.070840 0.163351 -0.802094 -0.560993 -0.218496 0.266565
 -0.831034 0.699503 -0.179889 -1.011836
 0.430336 0.333917 0.276990 0.023202 0.519719 -0.568560
 0.909640 -0.466165 -0.739116 0.160735
 0.367994 -0.963389 0.158938 -0.474918 0.692321 -0.663934
 -0.791958 -0.542801 0.620207 -0.808197
 -1.186396 -0.298362 0.571886 -0.319141 -0.300781 0.834413
 -0.285713 -0.493370 -0.739562 0.931413
 -0.941986 0.174351 0.777159 0.391117 0.310233 -0.216070
 -0.630436 0.037025 0.482231 0.269644
 -0.357729 -0.857912 -0.742514 -0.817627 -1.023082 0.564705
 -0.554891 -0.926618 0.736718 -0.313240
 -0.248375 0.307408 0.147777 0.087969 0.838803 -0.288084
 -0.560100 0.259130 0.610026 -0.807233
 -0.647187 -0.259216 0.471671 0.785001 0.139798 -0.170029
 -0.232439 -0.266674 -1.250269 0.450195
 -0.474672 0.827084 0.210791 -0.135320 -0.719628 0.667939
 0.906948 0.045485 0.347163 -0.109194
 1.547399 0.275682 0.288475 0.260190 0.041132 0.025974
 -0.047985 0.013011 -1.021868 -0.977792
 -0.211032 -0.453498 -0.951451 -0.015278 -0.561012 -0.265589
 0.531863 0.067146 -0.610323 -1.046850
 0.769789 -0.964489 0.472119 -0.296713 0.102355 1.026980
 0.922710 0.038599 0.610686 0.138571
 0.421568 0.546531 0.862450 0.361512 -0.144792 0.562609
 -0.753471 -0.109013 -1.461067 -0.578339
 1.007251 0.071899 -0.833274 -0.172144 -0.774923 0.504702
 -0.027266 -0.147188 -0.242898 0.041356
 1.441171 0.627201 0.054584 -0.427339 -1.316204 0.027579
 0.065096 -1.021183 -0.016677 -0.219800
 0.187152 -0.215512 -0.512761 -0.992741 -0.153881 -0.994530
 -1.100610 -0.570387 -0.601889 -0.318873
 1.347782 -0.210138 -0.253433 -0.325220 0.421802 -0.396122
 0.195505 -0.917572 0.598540 0.580319
 -0.028013 1.069686 0.779654 1.011575 0.508212 0.730162
 0.955521 -0.700511 -1.458425 -1.120590
 -0.573500 0.003982 1.111181 -0.172504 -0.318694 -0.957797
 -0.377070 0.577515 -1.226351 0.893687
 -0.176201 -0.111168 -0.531254 0.391972 0.595237 -0.673689
 0.815692 -0.685491 0.408676 -1.024247
 0.307414 -0.133296 -0.309397 -0.022472 -0.176239 -0.149549
 0.067351 0.531506 -0.455076 0.517335
 0.388613 0.037154 0.209694 -0.236981 -0.317801 -0.979943
 -0.549933 0.520306 -0.720037 -0.577649
 1.863607 -0.069145 0.891154 0.624496 0.203113 -0.772418

-1.247829 -1.003933 -1.141802 -0.294756
-0.140603 -0.011050 -0.180372 -0.983824 -0.502606 0.426398
-0.000484 0.174520 -0.022889 -0.157597
-0.090557 -0.623410 -0.909037 0.887225 -0.063196 -0.242529
0.562721 0.506873 -0.111744 0.020725
0.401597 -0.126068 -1.160244 -0.174082 0.516736 0.060420
0.871051 0.613144 0.106531 -0.460516
0.093251 -0.229145 -0.336990 -0.834517 0.447781 -0.363310
-0.265922 -0.330765 1.132744 1.016135
-0.928736 0.612305 -0.201531 -0.733867 0.871609 -0.094434
0.272833 -1.052732 -0.466353 0.825761
0.620701 -0.361398 0.367599 0.784988 -0.972825 0.256133
-0.631684 0.088232 -0.961979 -1.215079

Appendix G

Neural Net Training Data Set

Data from four subjects was gathered and preprocessed. Four second intervals (20 samples) were averaged and the reports were rounded to the nearest integer. The data vectors were then randomized.

Reports	GSR	EIG	ENG	Temp	Resp
7.000000	0.518760	0.145570	1.768390	0.155360	1.528615
7.000000	0.547680	0.099545	1.302490	0.144735	1.320615
9.000000	0.950840	0.947110	1.708720	0.155520	1.668480
7.000000	0.513830	0.153235	1.807800	0.149600	1.541070
10.000000	0.446610	1.454325	2.516425	0.790450	5.200265
7.000000	0.505890	0.152685	1.721330	0.165830	1.582380
7.000000	0.489250	0.163355	1.510080	0.160485	1.660815
7.000000	0.473170	0.175905	1.366060	0.138490	1.846230
1.000000	0.003725	0.038210	0.336710	0.119635	0.961565
10.000000	1.040480	1.019060	1.762990	0.141630	2.063680
7.000000	0.456170	0.209685	1.062990	0.145880	1.860390
2.000000	0.094410	0.058435	0.346445	0.225010	0.812190
1.000000	0.354885	0.090525	0.274120	0.228305	1.575375
5.000000	0.647725	0.167215	1.941555	0.538390	2.122760
7.000000	0.439665	0.236630	1.148395	0.196400	1.772860
7.000000	0.431215	0.293420	1.182060	0.182665	1.489460
1.000000	0.414515	0.100545	0.254465	0.221540	3.213755
8.000000	0.431710	0.289285	1.321585	0.188670	1.339005
8.000000	0.449955	0.287810	1.538065	0.227560	1.533145
8.000000	0.445530	0.344875	1.580310	0.240340	1.775115
9.000000	0.829550	0.806110	1.652140	0.113410	2.145436
8.000000	0.438765	0.343340	1.556095	0.225350	2.011940
8.000000	0.432675	0.343595	1.448545	0.230860	2.103275
10.000000	1.093775	1.084165	2.292080	0.093435	3.087030
8.000000	0.434600	0.339715	1.369850	0.231540	2.034860
8.000000	0.431750	0.404085	0.980945	0.239450	1.987240
1.000000	-0.006845	0.042805	0.308745	0.040150	0.686420
9.000000	1.000555	0.990545	3.042015	0.152400	1.914400
9.000000	0.183655	0.951380	1.778640	0.642955	2.318810
4.000000	0.483685	0.067825	0.339275	0.121900	1.208610
7.000000	0.526930	0.121035	1.670325	0.179470	1.309585
5.000000	0.636865	0.138120	2.511105	0.568175	2.463590
7.000000	0.451610	0.218820	1.168685	0.183240	1.866130
8.000000	0.432905	0.360370	0.870080	0.212250	1.834420
8.000000	0.433550	0.331700	0.891900	0.221650	1.796325
8.000000	0.430960	0.326640	0.966970	0.260570	1.768795
9.000000	0.835010	0.829175	0.916675	0.080695	1.881125
1.000000	0.361220	0.064700	0.397490	0.097835	1.377175
8.000000	0.426620	0.351335	0.910895	0.256680	1.731875
6.000000	0.110735	0.087605	2.156365	0.601170	1.908840
8.000000	0.412940	0.282210	1.047520	0.223640	1.764535

8.000000	0.412300	0.277225	1.090720	0.225015	1.699555
8.000000	0.416705	0.299015	1.131140	0.239425	1.478045
2.000000	0.082790	0.130425	0.349085	0.258355	1.649755
5.000000	0.486630	0.064520	0.372375	0.124830	1.240045
1.000000	0.097895	0.048705	0.379220	0.197115	1.040205
10.000000	0.472870	1.223070	2.393685	0.768820	4.966030
1.000000	0.287445	0.056070	0.323485	0.115360	1.465725
8.000000	0.442585	0.292430	1.441805	0.204275	1.505560
1.000000	-0.020015	0.010975	0.282500	0.054225	0.896040
8.000000	0.652385	0.599155	1.778665	0.230920	2.167110
1.000000	0.373320	0.196280	0.414305	0.173070	1.655080
8.000000	0.399160	0.253475	1.570635	0.262755	1.387265
2.000000	0.094645	0.104820	0.368925	0.215825	1.777750
8.000000	0.387985	0.264020	1.466535	0.254155	1.355655
8.000000	0.377700	0.244810	1.398090	0.259990	1.394730
8.000000	0.369810	0.262465	1.328925	0.257485	1.513820
2.000000	0.389930	0.064935	0.412870	0.073660	1.292075
9.000000	0.833375	0.815190	1.159995	0.118740	2.090735
8.000000	0.365675	0.286750	1.058665	0.257340	1.566390
6.000000	0.561355	0.519320	2.426606	0.175270	2.107910
1.000000	0.122820	0.050480	0.231485	0.019435	0.933060
7.000000	0.461105	0.192340	1.202505	0.145190	1.839820
8.000000	0.361995	0.283335	0.734525	0.308735	1.490160
8.000000	0.350195	0.276755	0.820190	0.313635	1.544265
9.000000	0.822800	0.792730	2.015140	0.139885	2.367855
2.000000	0.096760	0.086375	0.290515	0.225615	1.258170
3.000000	0.092715	0.144135	0.505775	0.450050	2.366950
6.000000	0.444400	0.401950	0.610855	0.137455	1.985400
1.000000	0.377555	0.196265	0.319440	0.205090	1.530985
9.000000	0.974490	0.959215	2.086455	0.147220	2.357465
6.000000	0.544090	0.494805	2.170640	0.142230	1.877465
3.000000	0.091725	0.053100	0.942900	0.379685	0.819545
8.000000	0.406700	0.292000	1.344055	0.252105	1.342485
7.000000	0.178340	0.851095	1.553520	0.653375	2.297815
3.000000	0.573715	0.087105	0.401570	0.431315	3.011960
1.000000	0.245680	0.044850	0.249685	0.104605	1.254825
8.000000	0.342635	0.267865	0.944850	0.305190	1.520610
8.000000	0.341280	0.200080	0.958645	0.341005	1.335810
2.000000	0.217905	0.095130	1.222675	0.111380	2.177305
9.000000	0.802545	0.777485	2.287335	0.121960	2.438220
9.000000	0.940265	0.892220	1.918775	0.161545	1.769760
2.000000	0.536305	0.149355	0.526530	0.374995	1.524465
2.000000	0.499415	0.124230	0.313585	0.313395	2.013035
1.000000	-0.047120	0.032680	0.525975	0.100810	1.107495
1.000000	-0.011235	0.010575	0.293060	0.059975	1.509570
10.000000	1.002695	0.999345	2.724455	0.139625	1.881170
6.000000	0.097935	0.098115	2.299010	0.604905	1.812280
1.000000	-0.034250	0.054330	0.328890	0.141970	0.904520
1.000000	-0.028560	0.018235	0.395070	0.068455	0.799940
3.000000	0.067030	0.090025	0.456765	0.346035	1.087910
1.000000	0.025060	0.079755	0.321575	0.086540	1.012370
2.000000	0.551155	0.135135	0.403910	0.321845	1.853580

3.000000	0.572030	0.085855	0.433705	0.390885	2.851285
9.000000	0.836080	0.833565	0.957420	0.115885	1.795625
1.000000	0.158895	0.081915	0.394530	0.141245	1.309530
3.000000	0.081355	0.090200	0.342125	0.339005	1.144300
3.000000	0.551810	0.071560	0.415640	0.407640	2.937840
6.000000	0.492755	0.421690	0.655265	0.158240	2.020350
9.000000	0.520140	0.223485	2.111790	0.733415	2.250555
2.000000	0.025760	0.032065	0.593195	0.103810	1.694795
2.000000	0.090165	0.043775	0.350455	0.220165	0.751795
3.000000	0.080345	0.075265	0.364390	0.312970	1.136925
5.000000	0.077785	0.140845	2.277590	0.555595	1.161415
3.000000	0.068350	0.125460	0.402515	0.287445	1.560060
9.000000	0.971620	0.954725	1.510035	0.147820	1.993205
9.000000	0.995845	0.979620	3.254215	0.135675	2.331675
10.000000	1.079255	1.048420	1.635525	0.117960	2.587435
10.000000	0.182205	0.920895	1.979865	0.665245	2.220140
9.000000	0.946070	0.870820	1.721535	0.163305	1.642770
7.000000	0.616855	0.560820	2.220215	0.338895	2.173950
1.000000	-0.018535	0.006430	0.143410	0.070985	1.129525
1.000000	-0.014910	0.029105	0.293615	0.048870	0.602105
1.000000	0.437415	0.088660	0.240635	0.248545	3.321950
1.000000	0.453320	0.092370	0.244555	0.276765	2.172670
1.000000	0.037390	0.051400	0.413700	0.244510	0.659910
5.000000	0.410300	0.372740	0.654590	0.170435	1.940205
8.000000	0.690460	0.643340	1.940515	0.211510	2.048885
7.000000	0.638345	0.128150	1.715000	0.628255	2.888031
8.000000	0.530955	0.228940	2.055225	0.714470	2.005900
1.000000	0.003320	0.040655	0.342185	0.119195	0.984210
6.000000	0.110480	0.439550	1.629155	0.648245	2.224650
1.000000	0.118335	0.044950	0.243440	0.031085	0.908245
1.000000	0.117495	0.042180	0.248510	0.022440	0.856220
1.000000	-0.058485	0.061090	0.289520	0.150700	0.898630
2.000000	-0.011370	0.044015	0.536850	0.127800	0.819405
1.000000	0.150475	0.055545	0.228745	0.004970	0.924235
1.000000	0.034630	0.086725	0.306820	0.078340	1.176340
1.000000	0.189230	0.070655	0.251725	0.027250	0.897150
5.000000	0.385490	0.346770	0.628010	0.175210	1.809220
1.000000	0.175160	0.068660	0.245020	0.029875	0.852735
1.000000	0.036565	0.080850	0.332190	0.069550	1.485530
1.000000	0.325055	0.067505	0.385650	0.091680	1.425620
3.000000	0.065075	0.135675	0.358430	0.294255	1.636110
6.000000	0.659460	0.131450	1.943940	0.610360	2.864595
9.000000	0.747850	0.704620	2.422885	0.147215	1.828715
1.000000	-0.050875	0.043300	0.444240	0.097970	0.951020
1.000000	0.368030	0.062885	0.419835	0.116585	1.258410
9.000000	0.961870	0.949720	1.585165	0.147945	1.695935
2.000000	-0.017670	0.041455	0.504350	0.105065	0.766150
10.000000	1.067145	1.037010	1.121815	0.100465	2.335120
2.000000	0.085175	0.097325	0.288570	0.204685	1.719555
2.000000	0.089300	0.055245	0.780910	0.368195	1.697060
2.000000	0.088270	0.054660	0.909980	0.394320	0.985650
1.000000	0.417545	0.096535	0.246620	0.225010	3.305170

1.000000	0.090700	0.041020	0.342120	0.229610	0.807005
6.000000	0.647905	0.151070	2.025305	0.581135	2.983390
3.000000	0.087115	0.155120	0.742940	0.476670	2.413005
9.000000	0.947595	0.945170	1.782960	0.154510	1.713465
10.000000	0.557515	1.058680	2.390930	0.758350	3.600456
1.000000	0.307095	0.064410	0.374185	0.112235	1.449830
1.000000	0.131875	0.043955	0.234030	0.006255	0.922250
2.000000	0.380095	0.063025	0.452715	0.103580	1.191105
2.000000	0.392345	0.065355	0.414800	0.081210	1.336390
8.000000	0.598805	0.103025	2.168325	0.668100	2.993035
10.000000	1.090110	1.077140	2.217785	0.120430	2.984140
10.000000	0.579870	0.681340	2.208765	0.758525	3.194745
3.000000	0.442630	0.056790	0.396880	0.095230	1.363315
9.000000	0.978520	0.969885	2.939780	0.169615	2.473450
1.000000	-0.005735	0.008945	0.226955	0.059540	1.457640
4.000000	0.466345	0.053100	0.349355	0.111010	1.517730
4.000000	0.080360	0.123595	1.820020	0.498960	1.267875
10.000000	0.425835	1.387145	2.581015	0.806695	4.734506
3.000000	0.068955	0.136950	0.346280	0.295990	1.596255
1.000000	0.171965	0.064910	0.249360	0.007970	0.917000
4.000000	0.487060	0.059770	0.353300	0.095820	1.335180
3.000000	0.079725	0.094540	0.410965	0.324790	1.158240
1.000000	0.000815	0.008555	0.227800	0.042585	0.436530
1.000000	-0.051335	0.038100	0.496260	0.083275	1.162970
4.000000	0.676565	0.134105	0.377365	0.477255	2.226130
1.000000	-0.005020	0.008725	0.203220	0.086190	1.411240
9.000000	0.947105	0.913530	2.144935	0.152180	1.809090
2.000000	0.376970	0.066840	0.361055	0.117035	1.147780
4.000000	0.487210	0.070510	0.391630	0.105750	1.224990
1.000000	0.136725	0.058275	0.242570	0.004925	0.931350
4.000000	0.487435	0.070375	0.362925	0.110770	1.222215
9.000000	0.721430	0.690440	2.326085	0.154165	1.913685
6.000000	0.108395	0.182220	1.702235	0.636005	2.138520
8.000000	0.691885	0.659340	1.982425	0.212285	2.046490
1.000000	-0.009025	0.009685	0.253005	0.043745	1.487560
6.000000	0.516005	0.445870	1.390470	0.159975	1.567160
9.000000	1.001815	0.995420	2.842570	0.160390	1.855480
2.000000	0.085620	0.098705	0.355650	0.228170	1.736315
1.000000	0.091350	0.051315	0.350975	0.238245	1.045430
1.000000	-0.019920	0.052725	0.347360	0.057820	0.712265
6.000000	0.530445	0.470400	1.980700	0.157710	1.568355
1.000000	0.387420	0.089730	0.256550	0.227785	3.061400
3.000000	0.068665	0.088660	0.431275	0.296100	1.345315
10.000000	1.082840	1.059080	1.824920	0.121555	2.742120
3.000000	0.574235	0.077985	0.407030	0.420715	2.994985
10.000000	1.020805	1.006310	2.488510	0.125175	1.644070
9.000000	0.796075	0.738995	2.678870	0.143945	2.331340
3.000000	0.095425	0.128805	0.671250	0.447355	1.972150
5.000000	0.075385	0.120515	2.242830	0.575015	1.693665
9.000000	0.861835	0.837240	0.832980	0.149450	1.828765
3.000000	0.099250	0.076415	0.964915	0.408950	1.064110
6.000000	0.135660	0.771455	1.680335	0.641535	2.281575

10.000000	0.179505	0.844155	2.223260	0.726710	1.997185
4.000000	0.341725	0.311755	0.568330	0.162180	1.825915
2.000000	0.090705	0.092290	0.277715	0.200335	1.683255
3.000000	0.082605	0.097600	0.350195	0.332185	1.141685
2.000000	0.101885	0.127340	0.359655	0.217675	1.714370
6.000000	0.415100	0.386380	0.652805	0.137745	1.928510
3.000000	0.078065	0.061480	0.366795	0.319885	1.700290
3.000000	0.077295	0.146740	1.017605	0.500705	2.327770
1.000000	0.016395	0.004115	0.045580	0.018115	0.305065
1.000000	-0.010960	0.059455	0.322090	0.058585	1.227840
1.000000	0.387850	0.153240	0.423920	0.177550	1.613300
9.000000	1.003555	0.985100	3.081695	0.135940	2.043155
1.000000	0.389950	0.196140	0.426325	0.183685	1.749505
4.000000	0.655790	0.101565	0.379310	0.469900	2.150940
5.000000	0.671275	0.158860	2.273740	0.562275	2.413180
9.000000	0.803925	0.759885	2.478925	0.141720	2.506955
8.000000	0.186830	0.903365	1.624040	0.641440	2.252875
1.000000	0.377255	0.194420	0.425165	0.209395	1.650880
1.000000	0.420160	0.089695	0.232985	0.231735	3.351730
1.000000	-0.055600	0.066230	0.311550	0.129550	1.052115
5.000000	0.084465	0.138645	1.895520	0.583975	1.171955
2.000000	0.570355	0.140365	0.462455	0.393045	2.599885
6.000000	0.641050	0.108755	2.568055	0.575875	2.608245
6.000000	0.664885	0.135595	2.330530	0.580720	2.949140
7.000000	0.630135	0.125160	1.835275	0.635390	3.192600
9.000000	0.975225	0.964125	2.590715	0.152305	2.529430
3.000000	0.083680	0.046030	0.380055	0.351230	1.773365
3.000000	0.065155	0.091330	0.449105	0.321355	0.996525
5.000000	0.074535	0.128290	2.185235	0.562795	1.304845
8.000000	0.581725	0.080575	2.233350	0.672665	3.084425
8.000000	0.538640	0.226795	2.152160	0.695530	2.426580
2.000000	-0.019655	0.037720	0.493220	0.098930	0.535400
2.000000	0.269430	0.145245	1.220470	0.110225	2.325415
8.000000	0.704445	0.675750	2.169590	0.189615	1.991650
2.000000	0.536305	0.172230	0.503870	0.342055	1.641890
5.000000	0.393725	0.360210	0.637430	0.180180	1.838830
1.000000	0.056800	0.089425	0.337095	0.078095	1.524375
8.000000	0.562220	0.179090	2.147790	0.675645	2.960695
1.000000	0.147125	0.073010	0.266315	0.008310	0.963535
1.000000	0.197975	0.049730	0.224460	0.086315	0.963700
2.000000	0.561165	0.173770	0.454555	0.331245	1.713755
3.000000	0.331640	0.243060	1.182650	0.110000	2.089025
1.000000	0.077850	0.051690	0.381920	0.259740	1.035650
3.000000	0.627010	0.093830	0.404090	0.443135	1.933940
3.000000	0.413610	0.057565	0.292620	0.079180	1.166515
7.000000	0.592160	0.539630	2.441100	0.288020	2.130785
10.000000	1.034880	1.011610	2.066820	0.132535	1.953940
3.000000	0.077380	0.112740	1.104080	0.497035	2.105095
1.000000	-0.033325	0.022650	0.387780	0.102970	0.921480
7.000000	0.620105	0.578785	1.886390	0.315570	2.157820
2.000000	0.090495	0.051275	0.577085	0.364945	1.727295
3.000000	0.099065	0.125185	0.852520	0.431130	1.444185

4.000000	0.086505	0.131930	1.847985	0.544510	1.230080
1.000000	0.051770	0.051895	0.402135	0.242740	0.976135
1.000000	0.045470	0.090310	0.334065	0.080130	1.544475
1.000000	0.008075	0.062415	0.342110	0.100110	1.010335
3.000000	0.346735	0.283755	0.649565	0.148790	1.914170
1.000000	0.009120	0.067075	0.305970	0.056710	1.452160
3.000000	0.080910	0.051090	0.355935	0.327130	1.781925
4.000000	0.646020	0.170250	1.468755	0.518280	2.181210
10.000000	1.014720	1.002935	2.671095	0.125825	1.663775
8.000000	0.675855	0.621120	1.651910	0.206515	2.095645
9.000000	0.952665	0.933920	2.161195	0.139700	1.771580
1.000000	-0.005330	0.046725	0.335405	0.133000	0.931345
7.000000	0.603720	0.089875	2.116185	0.656115	3.244525
1.000000	0.184830	0.070960	0.255475	0.045205	0.939105
1.000000	0.259470	0.049875	0.277045	0.108770	1.362255
1.000000	-0.024885	0.013920	0.328040	0.052255	0.591665
2.000000	0.384125	0.068385	0.392430	0.101490	1.120210
3.000000	0.421295	0.053660	0.307800	0.062885	1.174370
3.000000	0.445430	0.051915	0.331870	0.060560	1.342110
3.000000	0.449950	0.056635	0.354880	0.089680	1.413065
3.000000	0.437905	0.053700	0.440545	0.081960	1.354220
1.000000	0.092925	0.043025	0.366285	0.210230	1.013060
3.000000	0.444230	0.053400	0.440550	0.092220	1.328775
10.000000	0.487430	1.095310	2.345195	0.780360	4.307055
3.000000	0.464025	0.051630	0.410595	0.123595	1.387760
3.000000	0.396220	0.063570	0.283830	0.090380	1.281065
3.000000	0.462080	0.046955	0.394210	0.135830	1.463725
4.000000	0.455620	0.045645	0.370090	0.118160	1.466935
3.000000	0.604525	0.089055	0.407190	0.432210	2.059745
4.000000	0.457010	0.048655	0.361430	0.108970	1.501220
4.000000	0.489580	0.070810	0.376100	0.090185	1.224410
1.000000	-0.009990	0.019565	0.265755	0.048925	0.515105
10.000000	1.085880	1.068275	2.011615	0.102250	2.894845
9.000000	0.985810	0.974855	3.139615	0.138500	2.362060
9.000000	0.920915	0.849905	1.451620	0.170285	1.545295
1.000000	0.374440	0.147755	0.283305	0.222510	1.279045
5.000000	0.482470	0.052150	0.383175	0.106795	1.181625
2.000000	0.485725	0.108375	0.285965	0.301760	2.010130
5.000000	0.484470	0.046785	0.432985	0.100105	1.102175
5.000000	0.359515	0.330640	0.599415	0.180250	1.760330
9.000000	0.772220	0.719895	2.534395	0.128355	2.056000
10.000000	0.174460	1.134665	2.540435	0.730040	2.346290
5.000000	0.489610	0.045735	0.642260	0.104945	1.129560
5.000000	0.487165	0.048825	0.914045	0.093675	1.103525
4.000000	0.646880	0.174250	0.944520	0.499825	2.267965
10.000000	1.047060	1.027355	1.531175	0.129055	2.195435
6.000000	0.482940	0.054930	1.306245	0.089795	1.078600
6.000000	0.480000	0.066930	1.922970	0.115890	1.096470
1.000000	0.465845	0.096905	0.242195	0.290975	2.040140
1.000000	0.171755	0.059410	0.250685	0.040160	0.805000
6.000000	0.483575	0.073270	2.468700	0.145250	1.117680
4.000000	0.079725	0.127535	1.611865	0.491875	1.661605

6.000000	0.481580	0.071910	2.795660	0.132630	1.098540
6.000000	0.478190	0.067700	2.802005	0.118770	1.245855
6.000000	0.484580	0.066230	2.668700	0.122830	1.441060
6.000000	0.487120	0.069325	2.316955	0.147710	1.665260
7.000000	0.496445	0.085765	1.769295	0.173660	1.654810
2.000000	0.311395	0.195785	1.201505	0.094045	2.361100
2.000000	0.141225	0.047640	1.175350	0.105220	2.039140
7.000000	0.503475	0.092380	1.267110	0.173835	1.783340
1.000000	-.038210	0.027340	0.472880	0.112550	0.939465
9.000000	0.839155	0.822190	0.990290	0.102270	1.933445
2.000000	0.547465	0.158975	0.533985	0.357790	1.579690
7.000000	0.506590	0.091600	1.171310	0.156450	1.851910
7.000000	0.520665	0.095070	1.223475	0.148260	1.689695
7.000000	0.527285	0.088315	1.166135	0.147455	1.495400
4.000000	0.636420	0.151405	0.614410	0.495590	2.214160
7.000000	0.542475	0.078065	1.163895	0.126175	1.489940
7.000000	0.536800	0.124290	1.590025	0.180195	1.178375

Bibliography

1. Asystant. Software Documentation Manual. MacMillan Software Co., New York, NY, undated.
2. Czelen, W.E., M.D., Research Contractor, AFIT/ENG. Personal interviews. AFIT/EN, Wright-Patterson AFB, OH, December 1986 to March 1987.
3. Dhenin, G. Aviation Medicine. London:Tri-Med Books Limited, 1978.
4. Drylie, M. An Analysis of Physiological Data Related to Motion Sickness for Use in a Real-Time Processor. MS thesis, AFIT/GE/ENG/87D-16. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1987.
5. Fitzpatrick, D.G., M.A. Rogers, and R. Williams. Computerized Biophysical Data Acquisition System for Motion Sickness Studies. MS thesis, AFIT/GE/GCS/ENG/84D-30. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1984.
6. Gaudreault, P.J. Motion Sickness: A Study of its Effects on Human Physiology. MS thesis, AFIT/GE/ENG/87D-20. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1987.
7. Gillingham, K. and J. W. Wolfe. "Spatial Orientation in Flight," Fundamentals of Aerospace Medicine, edited by Roy L. DeHart. Philadelphia:Lea and Febiger, 1985.
8. Graybiel, A. and others. "Diagnostic Criteria for Grading the Severity of Acute Motion Sickness," Aerospace Medicine, 39: 453-455 (May 1968).
9. Hartle, Dana R. A Motion Sickness Prediction Model and System Description. MS thesis, AFIT/GCS/ENG/86D-3. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1986.
10. Jarvis, N.R. and C.T. Uyeda. An Analysis of Potential Predictive Parameters of Motion Sickness using a Computerized Biophysical Data Acquisition System. MS thesis, AFIT/GSO/ENG/85D-1. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1985.

11. Lackner, J. R., Ph.D and Ashton Graybiel, M.D. "Etiological Factors in Space Motion Sickness," Aviation, Space, and Environmental Medicine, 54: 675-681 (August 1983).
12. Levey, R. A. and others. "Biofeedback Rehabilitation of Airsick Aircrew," Aviation, Space, and Environmental Medicine, 52: 118-121 (February 1981).
13. Lippman Richard P. "An Introduction to Neural Nets," IEEE ASSP Magazine, (April 1987).
14. Matsnev, E. I., and D. Bodo. "Experimental Assessment of Selected Antimotion Drugs," Aviation, Space, and Environmental Medicine, 55: 281-286 (April 1984).
15. McPherson, Michael R. A Collection and Statistical Analysis of Biophysical Data to Predict Motion Sickness Incidence. MS thesis, AFIT/GCS/ENG/86D-14. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1986.
16. Miller, Robert D. Motion Sickness: A Study of its Etiology and a Statistical Analysis. MS thesis, AFIT/GCS/ENG/86D-2. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1986.
17. Royal, L. and others. "Motion Sickness Susceptibility in Student Navigators," Aviation, Space, and Environmental Medicine, 55: 277-280 (April 1984).
18. Warwick-Evans, L.A. and others. "Electrodermal Activity as an Index of Motion Sickness," Aviation, Space, and Environmental Medicine. 58: 417-423 (May 1987).

VITA

Captain Edward L. Fix was born the son of Paul and Ruth Ann Fix on 15 May 1952 in Vinton, Iowa. He graduated from Washington High School in Vinton in 1970 and attended Iowa State University in Ames, Iowa, earning a Bachelor of Science degree in Electrical Engineering in March, 1975. He attended Officer Training School and received his commission in January, 1979. After undergraduate pilot training at Reese AFB, Texas, He was assigned to Minot AFB, ND. There he served as a B-52 copilot and aircraft commander until May, 1986 when he was assigned to AFIT's School of Engineering at Wright-Patterson AFB, Ohio.

Permanent Address: 1124 Grove Hill Dr.
Beavercreek, OH 45385

A100000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/87D-18		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering			6b. OFFICE SYMBOL (If applicable) AFIT/ENG		5. MONITORING ORGANIZATION REPORT NUMBER(S)
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright Patterson AFB, OH 45433			7a. NAME OF MONITORING ORGANIZATION		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION			8b. OFFICE SYMBOL (If applicable)		7b. ADDRESS (City, State, and ZIP Code)
8c. ADDRESS (City, State, and ZIP Code)			9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
11. TITLE (Include Security Classification) MOTION SICKNESS: QUANTITATIVE, ALGORITHMIC MALAISE INDICATION IN REAL TIME (Unclassified)			10. SOURCE OF FUNDING NUMBERS		
12. PERSONAL AUTHOR(S) FIX, EDWARD LEE, B.S., CAPT, USAF			13a. TYPE OF REPORT MS Thesis		
13b. TIME COVERED FROM _____ TO _____			14. DATE OF REPORT (Year, Month, Day) 1987 December		15. PAGE COUNT 107
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Motion Sickness, Aerospace Medicine, Aviation Medicine, Physiologic Monitoring, Biofeedback		
06	10				
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
THESIS CHAIRMAN: Matthew Kabrisky, PhD. Professor of Electrical Engineering Department of Electrical Engineering (ABSTRACT ON BACK) Approved for public release: IAW AFR 190-17 22 JUL 88 1. Approved for release by AFIT/ENG/87D-18 2. Approved for release by AFIT/ENG/87D-18 3. Approved for release by AFIT/ENG/87D-18 Wright Patterson AFB OH 45433					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Matthew Kabrisky, PhD., Electrical Eng			22b. TELEPHONE (Include area code) (513) 255-5276		22c. OFFICE SYMBOL AFIT/ENG

Block 19. Abstract

Physiological data were collected on human volunteers to study the effects of motion sickness. Data were analyzed and correlated using least squares curve fitting and other statistical methods that are described. An equation is developed that relates five separate physiological signals to subjective motion sickness.

A computer program is presented and described which takes the physiological signals in real time and computes the motion sickness. A pattern recognition type of approach which uses a neural net is presented and discussed as an alternative to the equation model. The two models are compared.

END
DATE
FILMED
MARCH
1988
DTIC