MODEL-BASED SCIENTIFIC DISCOVERY: A STUDY IN SPACE BIOENGINEERING

by
Nicolas Groleau

I.T.P.E., Ecole Nationale des Travaux Publics de l'Etat, 1986

M. S., Massachusetts Institute of Technology, 1989

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
in Intelligent Engineering Systems

at the

Massachusetts Institute of Technology

September 1992

Signature of Author _____
Department of Civil and Environmental Engineering
August 17, 1992

Certified by _____
Professor Laurence R. Young, Department of Aeronautics and Astronautics
Thesis Co-Supervisor

Certified by _____
Professor Peter Szolovits, Department of Computer Science
Thesis Co-Supervisor

Certified by _____
Professor Jerome Connor, Department of Civil and Environmental Engineering

Certified by _____
Professor Duvurru Sriram, Department of Civil and Environmental Engineering
Department Reader

Accepted by _____
Professor Eduardo Kausel
Chairman, Departmental Committee on Graduate Studies

MODEL-BASED SCIENTIFIC DISCOVERY: A STUDY IN SPACE BIOENGINEERING

by
Nicolas Groleau

I.T.P.E., Ecole Nationale des Travaux Publics de l'Etat, 1986

M. S., Massachusetts Institute of Technology, 1989

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
in Intelligent Engineering Systems

September 1992

Abstract

This thesis describes a novel system that modifies the theory contained in a model of the normal human orientation system. This system, called MARIKA, demonstrates automated scientific discovery in an actual scientific domain through techniques adapted from diagnosis and design. MARIKA comprises a simulation module, a constraint propagation module and a model revision module. The model is captured in a simulation environment that produces time-varying signals. The model parameters are represented as constrained range variables, and the input, output and intermediary signals are segmented in time and approximated by linear combinations of a set of four simple shapes adequate for the vestibular domain. The model linear differential equations and boundary conditions are transformed into constraints on the curve fit parameters and the model parameters. Some extensions are also provided to cover simple non-linear cases and steady state conditions. Clinical data are abstracted using the shapes predicted by the model. These data are then compared to simulation predictions by propagating the constraints relating the model and the curve fit parameters. In case of contradiction, the model is modified by either extending the range of model parameters to pathological values or altering the structure of the model according to pre-indexed methods.

MARIKA correctly models normal vestibular data and several end-organ and nervous processing defects. MARIKA demonstrates the synergy of diagnosis and design techniques, qualitative and quantitative representation, and modeling, simulation, and artificial intelligence for a routine form of automated scientific discovery. MARIKA's set of shapes could readily be enhanced to be used to perform model-based theory refinement on a variety of linear domains. These encouraging results could lead to a useful clinical vestibular tool and to a space vestibular adaptation scientific discovery system.

Thesis Co-supervisor: Dr. Peter Szolovits
Title: Professor of Computer Science

Thesis Co-supervisor: Dr. Laurence R. Young
Title: Professor of Aeronautics and Astronautics

Dedication

I read my share of theses while working on mine. I always started by reading the dedication and acknowledgments pages. I often found them corny and exaggerated. I am lucky enough to find myself in the same strange but unavoidable position.

In dedicating this work I have only one regret. Somehow it might not justly reflect the quality of the support I received while doing it.

I would like to dedicate this thesis to my wife Patricia who suffered every bump along the way and took close to six years of leave from her life to let me accomplish one of my old dreams. I remember a newspaper clip she showed me in 1985 in a beatup suburb in central France, which described a pretty good engineering school in Massachusetts. Well, you can put it in the scrap book now and turn to the next page.

La lune est douce,
La lune est blanche
Ce soir.
Tu es la source
Où je m'épanche,
Miroir
Où se reflètent
Les mots désirs
Pour moi,
Qui ne souhaite
Que de mourir
Pour toi.
Amour fidèle,
Corps voluptueux,
Mon âme,
Ca tu es celle
Que je veux
Ma femme.

To you...

Acknowledgements

There are many people to thank, for the journey is long and I couldn't have traveled alone. I've tried my best to include everyone in a mostly random order.

Thanks to Don Kaiser for coming at the right moment and teaching me so much about the outside world. Yes...Remember kids, there is an outside world! Thanks to him as well for simply stating "In my mind, you certainly have the necessary attributes for getting your Ph.D., just make sure [they do] not convince you otherwise!" Many thanks too for reviewing the text of the thesis and helping make it into readable English.

Thanks to Pete Szolovits for his earnest and just evaluations. I didn't always like it but I knew all along it was for the best. Thanks for filling up a committee member spot when it was freshly deserted. Thanks for thinking of me when [PI] got started.

Thanks to Larry Young for giving me the opportunity to be introduced to the space program. Thanks also for constantly providing that element of uncertainty that at times felt like a yo-yo ride between heaven and hell. It was for the best? Right?

Thanks to Jerry Connor for providing rock solid encouragement in all circumstances.

Thanks to Duvurru Sriram for his continued role in all my years at MIT. I appreciate his editorial help and advise during both my theses. He gave me the opportunity to come to MIT in the first place.

Thanks to the MVL support staff for making it a livable place. Some conversations were long, some just a wink at the outset of a meeting but having a family at my home away from home was a blessing.

Thanks to Jim for teaching me all the hardware stuff I know and sharing some of his years of experience and wisdom of life.

Thanks to Beverly for making red tape what it is supposed to be, a little information, a signature and your on your way. Thanks too for unpredictable but insightful conversations on the widest variety of subjects (I meant patients).

Thanks to Kim Tseko for persevering in her role of dictatoress.

Thanks to Sherry for holding the house together and providing such example of a dedicated lab member. By choice I didn't take advantage of her culinary talents and I know I missed a lot.

Thanks to the MVL students, the old and the young generation: Keoki, Dave, Jock, Glenn, Valérie, Juan, Karla, Scott, and Ted. Your youthful enthusiasm made the passage of time more perceptible. Texas and southern California will hold memories of SLS-1 and hopefully SLS-2 will be reason for reunion. By the way Valérie, you know you can't graduate until a fellow French student joins MVL, and thanks for the Simulab help. Particular thanks to Keoki for excellence, to Dave for being a spontaneous friend, for a beery kind of pool game, and for his MATLAB legacy, to Jock for almost Mount Whitney kinds of things, and to Scott and Ted for MATLAB help.

Thanks to Dr. Dan, and Dr. Mark for showing the path.

Thanks to Dan Merfeld for being patient enough to teach me what the vestibular system is really made of and opening the doors of modeling. Being his worst student, the rest of his teaching career should be a breeze.

Thanks to Dava Newman for racing along with me part of the way and staying ahead just far enough.

Thanks to Lyman Hazelton for breathing a different wind into [PI] and being a different type of boss. He gave me the elbow room I needed for the last sprint, he unwedged me at critical points of my research and spent numerous hours reviewing the thesis document. Thanks for the things he does and for all the things he leaves for others to do. Long life to Scuttle! Upward, forward and onward...

Thanks to the rest of the [PI] team. In particular thanks to Irv Statler for initiating me to the art of perseverance in argumentation, to Michael Compton for teaching me that without PR we would all be working on OJECTS, to Rich Frainier for teaching me the other way to do software: the organized way. Thanks to him too for optimism, patience and always expecting more. Particular thanks go to Silvano Colombano who acted as a remote boss, an adviser, a friend and most of all provided the ultimate motivation: a flexible deadline and concrete post-graduation plans.

Thanks to Don Rosenthal for pointing me to LabVIEW even though this is the only mention of the software in this document. It made my RA so much more rewarding. Thanks also for a warm welcome and two great working trips in the Sierras. Thanks finally for hiring Silvano in the first place.

Thanks to the members of the AI Research Branch at NASA Ames Research Center who attended a talk I gave on an earlier version of this work and kicked me back in the right direction. The world will have to wait for a solution to the mystery box...

Thanks to my daughters Nastassia and Marika after whom I named some of the work I did over the first few years of their lives. They provided a constant reminder of what my priorities ought to be. I hope somehow those years of sacrifice and roller-coaster ride will pay back.

Thanks to Robert, Enock, Glenn, Pasquale, J-M, TB, NY, CN & ED and others for rounder edges.

Thanks to the Westgate crowd for Volleyball, barbecues and sharing the playground, rain or shine.

Thanks to Pascale and Bruno for a very long Saturday afternoon and a small but full treasure chest from the mainland.

Thanks to the IM soccer, Cosmos and the Aero & Astro team for providing two championships and some of the most childish frat 'baloney' I ever witnessed first hand.

Thanks to Véronique and Lois for being their for my wife all those years and for the whole family during Marika's crucial first semester in Fall 1989.

Thanks to my brother Pierre for helping me prove to myself that there is more than one path to fulfillment.

Thanks to Pierre Haren for teaching me what AI stands for and opening the door to MIT.

Thanks to Kim and Aziz for the Celtics tickets, and thanks to Debbie and David for the Patriots tickets.

Thanks to my hair for holding on while I was pulling on it and my head was spinning. It instead simply decided to turn white of fright.

Thanks to Conrad Wall III for providing wall to wall references (Don's pun), and let Gordon Arekawa and Dave Balkwill spend some of their time canning some data for me.

Thanks to Dr. Chuck Oman for paying so much attention to the mailboxes that he didn't notice the lab was running out of teaching faculty and doctoral students. Not that it makes a difference, nut it allowed me to slip by unhindered...

This just in time thesis was brought to you by grant NASA grant #NCC2-5705, with additional funding from Boeing contract #L1452-JJK91-786.

Biography

Nicolas Groleau was born in Neuilly-sur-Seine, a Parisian suburb, in early 1964. However, his parents moved to the French Riviera by the time he was four and that's where he likes to call home.

He obtained his Civil Engineer's degree from Ecole Nationale des Travaux Publics de l'Etat in Lyon, France, in June 1986, with a minor in Computer Science. After a year of research on multi-expert systems at INRIA, Nicolas started his Masters in Civil Engineering at MIT. He graduated in February 1989 with a thesis focusing on a Blackboard architecture for Design.

Switching gears, he started his inter-departmental doctoral work at the Man-Vehicle Laboratory in the Aeronautics and Astronautics Department where he developed an interest for the vestibular system, image analysis, and Macintosh-based data acquisition and analysis.

His most pressing goal is to start his career before he reaches thirty and begin saving for his kids' college.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

To reduce experience to order is a task older than science, the ancient goal of the philosophers. But to augment experience is the task that engages the artist, the artisan, the traveler, and indeed every child, who is the learner in us all. It was out of the slow fusion of the two that science appeared glowing in the crucible of history, and its composition is still visible in its affinities: science owes heavy debts to philosophy, art, craft, exploration, and childhood. [Morrison and Morrison 1987]

It is impossible to dissociate language from science or science from language, because every natural science always involves three things: the sequence of phenomena on which the science is based, the abstract concepts which call these phenomena to mind, and the words in which the concepts are expressed. To call forth a concept, a word is needed; to portray a phenomenon, a concept is needed. All three mirror one and the same reality. [Lavoisier 1789]

I. Introduction

As put so well in [Langley et al. 1987], I believe the process of scientific discovery is a domain worth studying:

> "The story of scientific progress reaches its periodic climaxes at the moments of discovery...
> However romantic and heroic we find the moment of discovery, we cannot believe either that the events leading up to that moment are entirely random and chaotic or that they require genius that can be understood only by congenial minds. We believe that finding order in the world must itself be a process impregnated with purpose and reason. We believe that the process of discovery can be described and modeled, and that there are better and worse routes to discovery—more and less efficient paths."

Scientific discovery has been a focus of recent Artificial Intelligence (AI) developments. The fundamental argument is that intelligent systems should autonomously discover a large portion of the knowledge they require. It is indeed becoming clearer that the amount of knowledge needed for powerful specialized systems or for general purpose systems cannot be efficiently coded by hand. The fundamental question addresses what mechanisms will provide a computer system with such learning or discovering abilities. One natural exploratory path is to have the computer mimic the techniques of human scientific discovery as it is often well documented.

> "...a few hundred rules in a computational system can only provide a sketchy model. Nevertheless, what is attractive about computational modeling of discovery is that it provides us the insights on the reasoning behind the critical decisions taken by the scientists during the course of their research. A computational model also provides a panoramic view of the process of research and may even help to develop a methodology (or

methodologies) for better strategies for scientific research in different fields." [Kocabas 1992]

Some seminal works in automated scientific discovery include the early system AM [Lenat 1977], and the BACON programs and its successors [Langley et al. 1987, Karp and Friedland 1989, Kulkarni 1987, Kulkarni and Simon 1988, Langley and Nordhausen 1986, Valdés-Pérez 1990, Zytkow 1987]. Following these investigations, a series of works retracing historical discoveries, many being demonstrated in the chemistry domain, were developed. Though not all researchers agree on the exact bounds of scientific discovery, theory formation is clearly at the core of the domain. As scientists advance in their understanding of a domain, they develop models that are sets of hypotheses supported to varying degrees by experimental facts. With the risk of oversimplifying, as the knowledge acquired by the scientists increases, the theory formation effort progresses from the formation of qualitative laws through quantitative laws to the unification of theories. In a way parallel to engineering design, scientific discovery can be innovative to routine. Intuition and genius are words often used to characterize the first phase of scientific discovery. The establishment of numerical laws and the explanation of the phenomena from first principles are difficult tasks to automate. This is innovative science. My work focuses on the more routine aspects of discovery. Indeed, my system includes a countable number of model alteration techniques. They can be combined to produce non-trivial model variations, but the space of models explored can be outlined *a priori*. In this thesis I present a system that performs model-based scientific discovery. In accordance with [Hawking 1988]:

> "I shall take the simple-minded view that a theory is just a model of the universe, or a restricted part of it, and a set of rules that relate quantities in the model to observations that we make. It exists only in our minds and does not have any other reality (whatever that might mean). A theory is a good theory if it satisfies two requirements: It must accurately describe a

> large class of observations on the basis of a model that contains only a few arbitrary elements, and it must make definite predictions about the results of future observations."

I believe model-based theory formation is an important technique, because the traditional scientific method is to observe a phenomenon, make a hypothesis and then prove or disprove the hypothesis, at least from a statistical point of view. When trying to prove or disprove a hypothesis, the scientist is trying to test a model of the domain, a single hypothesis model.

The majority of discovery systems in AI use well documented historical discoveries or toy problems. While it is clear that the latter may be unconvincing because of the conceptual leap between the perfect small world of a toy problem and an actual scientific domain, the former has its own drawbacks. Proving that somehow the solution was not accidentally coded into the system is difficult, because the computer scientist clearly knows the answer *a priori*. Moreover, such re-discovery systems rarely prove their usefulness by being tested in a scientific domain in its current development stage. Another argument is given in [Zytkow and Baker 1991]:

> "Typically, discovery systems have relied either on user input or on simulation. In the former case. the data are severely limited in numbers; in the latter, they are idealized, lacking many real-world characteristics."

I originally chose the adaptation of the vestibular domain to weightlessness conditions as my scientific field of experiment. Providing selective stimuli to a single sensor or suppressing a single signal is often impractical or unethical for human subjects. However, the micro-gravity environment provides altered conditions under which the signals due to gravity are completely eliminated. The experiment I originally chose for this investigation is performed both on the ground and in space. Comparison of these two

experimental conditions provides insight into the role the gravity receptors play in spatial orientation. My initial goal was to design a system that could predict the behavior of the vestibular system for a set of astronauts in micro-gravity from baseline ground data using a vestibular domain model. However, the data gathered during the Spacelab Life Sciences 1 (SLS-1) space shuttle mission are inconsistent with the current views of the vestibular system and cannot be predicted by incrementally modifying the current model. Because further space data will be difficult to obtain, I have shifted to a more reliable data set from clinical vestibular research.

In this thesis I describe a system that modifies the theory contained in a model of the normal human spatial orientation system. Each parameter of the model is represented as a range of possible values for normal subjects. As the examination of an individual progresses through a series of clinical tests, modifications to model parameters are made within the constraints of normality. If the individual is a normal subject, the system constrains the set of parameters describing the vestibular system of that individual. If the individual has a response indicating a pathological case, the system is unable to find a consistent set of parameters. The system then modifies the structure of the model according to the discrepancies and tunes the new model. The final result is a model structurally different from that of a normal subject but that represents the subject's pathological vestibular system. This system is a step towards automation of model-based theory revision. The choice of pathological vestibular research simplifies knowledge engineering and validation. The field has been studied over a longer period of time than vestibular adaptation to weightlessness and data are more abundant. Moreover, the pathological subjects I model have well defined medical conditions that are used to evaluate the system's output. Nevertheless, the techniques developed here are readily applicable to help solve the original model development task. However, additional structural modification methods, such as the addition of a new parameter or the

introduction of a new signal, will be necessary to solve the original task; yet they are not addressed in this thesis.

After this brief introduction, I present an overview of relevant AI research done in scientific discovery in Chapter 2. The relevance to the work presented here is outlined. Chapter 3 introduces the vestibular domain and the human orientation modeling approach I used for this thesis. In Chapter 4 I describe my approach to model-based scientific discovery. I present my initial goal of space vestibular adaptation discovery system before justifying the redirection towards the clinical environment. Each step of the system from initial tuning to selection of the most parsimonious model is discussed. Chapter 5 details working examples of my system called Model Analysis and Revision of Implicit Key Assumptions (MARIKA). These include data from normal subjects as well as various types of pathological cases in both rotating chair and off vertical axis rotation, known as "barbecue spit" conditions. I conclude in Chapter 6 by assessing contributions of this work as well as outlining its limitations. I also sketch directions for future work.

CHAPTER 2

Scientific Discovery

The task of science is both to extend the range of our experience and to reduce it to order, and this task presents various aspects inseparably connected with one another. Only by experience itself do we come to recognize those laws which grant us a comprehensive view of the diversity of phenomena. As our knowledge becomes wider, we must even be prepared therefore to expect alterations in the point of view best suited for the ordering of experience. [Bohr 1929]

II. Scientific Discovery

II.1. Introduction

When working on routine science and trying to add even an infinitesimal brick to the edifice of science, one runs into the problem of the nature of scientific theories. The issue here is how to prove a scientific theory correct. As opposed to mathematical theories where theorems are proven logically, a scientific theory cannot be proven. It can only be disproved if one of its predictions is contradicted by an experiment. A theory becomes closer and closer to being considered true as the number of minor theories built upon it (that use it as an underlying principle) increases. The edifice of science is built from first principles that are themselves small theories that serve as a sound foundation for the rest of science. The truth value of a theory increases with the truth value of all theories that use it as part of their argumentation. Typically these building blocks have survived the test of time. In the scientist's mind they slowly drift from long lasting theory to undeniable truth.

> "Any physical theory is always provisional, in the sense that it is only a hypothesis: you can never prove it. No matter how many times the results of experiments agree with some theory, you can never be sure that the next time the result will not contradict the theory. On the other hand, you can disprove a theory by finding even a single observation that disagrees with the predictions of the theory. As philosopher Karl Popper has emphasized, a good theory is characterized by the fact that it makes a number of predictions that could in principle be disproved or falsified by observation. Each time new experiments are observed to agree with the predictions the theory survives, and our confidence in it is increased; but if ever a new observation is found to disagree, we have to abandon or modify the theory.

> At least that is what is supposed to happen, but you can always question the competence of the person who carried out the observation.
> In practice, what often happens is that a new theory is devised that is really an extension of the previous theory." [Hawking 1988]

Scientific discovery has been a focus of recent Artificial Intelligence (AI) developments. With the risk of oversimplifying, as the knowledge acquired by the scientists in a domain increases, the theory formation effort progresses from the formation of qualitative laws to the unification of theories. Scientific theory formation can be broken into several tasks requiring different techniques as the overall knowledge developed for the domain increases. Several levels of theory formation are described in Table 1.

No.	Level of Theory Formation	Amount of Work
1	Qualitative Laws	Large
2	Quantitative Laws	Large
3	Partial Hypotheses (Explanations from first principles)	Moderate
4	Complete Theories from Partial Hypotheses	Moderate + This Work
5	Generalization of Local Theories	Large
6	Unification of Theories	None

Table 1: Levels of Scientific Discovery.

The first level of theory formation requires the scientist to notice patterns in the data at the qualitative level. Starting from numbers, the scientist looks for patterns and regularities in order to come up with qualitative laws describing the phenomena that one can observe. The direction of variation of numeric terms are compared and influences between those terms are sought.

These qualitative findings can be transformed into quantitative statements by combining related numeric terms and looking for other qualitative or numeric relations. For example,

if variable X and variable Y increase simultaneously, the ratio $\frac{X}{Y}$ may be constant. A numerical law of the form $\frac{X}{Y}$ = constant is discovered. Interesting work has been done in that area with programs such as BACON, [Langley et al. 1987], Fahrenheit [Zytkow 1987] and IDS [Langley and Nordhausen 1986].

More recently, a significant effort has been started that deals with discovery in large databases and is reviewed in [Piatetsky-Shapiro and Frawley 1989]. However, typical scientific discovery is usually poor in data and cannot use such data hungry techniques.

> "We point out that database miners cannot count on the same quality of data as scientists can. Database miners must content themselves with sparse, static data, because they cannot count on experiments and data refinement available in scientific data collection." [Zytkow and Baker 1991]

Scientific discovery usually involves interaction between data gathering and data modeling or data explaining in one way or another. In contrast, large database discovery cleanly separates the two activities in time.

> "In a database, the data are fixed and sparse. The number of records is fixed and no other observational attributes can be introduced." [Zytkow and Baker 1991]

Once the numerical laws have been established, the scientist can look for explanations to why these laws hold. According to Webster, a law is a statement of an order or relation of phenomena that so far as is known is invariable under the given conditions. On the other hand, an explanation shows the reason for or the cause of. The second phase of scientific discovery is to find the underlying principles that explain those laws. This is complementing the 'how' can these data be modeled by a 'why' can they be modeled that

way. Typically, elementary scientific principles can be hypothesized to hold in the domain studied that lead to the observed laws.

> "Une théorie biologique n'a de sens que si elle part de l'observation des objets naturels et y retourne rapidement. Une première manière d'éprouver sa solidité consiste à examiner la plausibilité des mécanismes élémentaires sur lesquels elle se fonde." [Changeux 1983]
> *Biological theories do not make sense unless they are rooted in the observation of natural objects and rapidly return to them. A straightforward way to test their strength is to examine the plausibility of the elementary mechanisms on which they are founded.*

These principles can be very basic or themselves built from lower level principles. However, great discoveries often require the introduction of a new principle. As these clusters of explanations are developed, partial hypotheses are formed. Some alternatives usually exist at various levels as the overall compatibility of the locally mapped knowledge hasn't been established. Discovering underlying principles can actually be done in two ways. One can consider the laws and try to infer which principles could produce such laws. Conversely one can hypothesize a particular explanation, calculate the nature of the law it would produce and compare it to the observed law. This is the way the scientific method always prefers. The predicted results are compared to the actual data and the match is statistically evaluated. The two techniques are also used interactively. The differences between the predictions and the data are left to be further modeled. The first pass determines the first degree law and subsequent passes allow the description of higher order terms.

The next level of theory formation formulates more complete theories from the local hypotheses. This stage of scientific discovery takes place when a mathematical model has evolved from previous research. Certain parts of this model are likely to be well

established, while others are more hypothetical. Model-Based theory formation can then be performed, as I am proposing in this thesis. The purpose of this technique is to build on the existing model by confirming it, modifying sub-hypotheses within it or refining it incrementally as new data are gathered. This is particularly useful for domains where obtaining each new data point is very costly or time critical, such as space research, biological recordings [Schement, and Hartline 1992], clinical settings, etc.

Finally, large domains will see the coexistence of a number of widely accepted and broad theories. As in Physics, such theories will ultimately have to be unified in order to smooth out arbitrary validity boundaries.

> "The eventual goal of science is to provide a single theory that describes the whole universe." [Hawking 1988]

This issue is currently at the crux of theoretical quantum mechanics where scientists are working (with significant success) on the unification of the four universal forces that can be seen as four complementary theories. The discovery of the unification of all these forces into one will probably the greatest accomplishment of scientific discovery. It will explain all physical phenomena by a single equation.

> "I still believe there are grounds for cautious optimism that we may now be near the end of the search for the ultimate laws of nature." [Hawking 1988]

It is, however, clear that despite the immense esthetic value of such a universal theory, routine science and consequently engineering need more practical tools with which to work. Therefore, broad theories are specialized into smaller ones that have a more narrow range of validity but that encompass a larger number of constraints, compiling a backward copy of scientific progress into a usable program.

> "It turns out to be very difficult to devise a theory to describe the universe all in one go. Instead, we break the problem up into bits and invent a number of partial theories. Each of these partial theories describes and predicts a certain limited class of observations, neglecting the effects of other quantities, or representing them by simple sets of numbers. It may be that this approach is completely wrong. If everything in the universe depends on everything else in a fundamental way, it might be impossible to get close to a full solution by investigating parts of the problem in isolation. Nevertheless, it is certainly the way we have made progress in the past." [Hawking 1988]

As I have just outlined, scientific discovery comprises many tasks, and in a way parallel to engineering design, these tasks can cover the spectrum from innovative to routine. My work focuses on the more routine aspects of the discovery process. Indeed, a large portion of the scientific effort consists of slowly pushing the validity of established theories in slightly novel ways. The system I present here makes small modifications to the vestibular system model in order to fit individual data as they are acquired. Similarly, a system working on vestibular space adaptation would incrementally modify the model of the vestibular apparatus to reflect the data set as it grows with data acquired in micro-gravity.

Other scientific discovery systems have been developed but do not address the level of discovery I just described. It seems therefore useful to review some of these systems and their relation to the work presented here.

II.2. Scientific Discovery Systems

II.2.1. Introduction

Some seminal works in scientific discovery include the early system AM [Lenat 1977], and the BACON programs [Langley et al. 1987]. Following these investigations, a series of works, many being demonstrated in the chemistry domain, were developed to retrace historical discoveries [Karp and Friedland 1989, Kulkarni 1987, Kulkarni and Simon 1988, Langley and Nordhausen 1986, Valdés-Pérez 1990, Zytkow 1987]. Table 2 shows some of the works mentioned above with a brief description. The systems most relevant to this thesis are discussed in the following sections[1].

II.2.2. AM

AM is a heuristic-driven elementary mathematics and set theory discovery system. The system doesn't discover concepts to explain observations, but simply explores to map out

---

1 COPER applies the model-driven approach to numerical discovery. The system's generator is knowledge rich. It uses dimensional attributes to constrain the generation of theoretical terms. Once its set of theoretical terms is complete, the system performs search in a space of polynomial functions to summarize the data.

ABACUS represents the scope of laws either as symbolic conditions or as simple maxima and minima on the values of numeric terms.

Unlike BACON, GLAUBER is presented with the entire set of data it must analyze. Therefore, it searches only a space of laws and concepts but no data space. The data it processes is entirely symbolic and GLAUBER generates qualitative laws only.

DALTON performs heuristic search in a space of molecular models which must explain chemical reactions. It can consider many reactions consecutively and intelligently backtracks if some later reaction requires a model revision.

STAHL is a GLAUBER-like program specialized in chemistry. Its initial state is a set of reactions and it tries to distinguish elements from compounds and to determine the elemental composition of each compound.

Rather than stating the scope of a law as simplistic independent values, FAHRENHEIT specifies these limits as another set of numerical laws using new theoretical terms.

mathematically interesting concepts. AM starts with 115 basic elements such as sets, lists, and elementary relations such as equality. AM has successfully rediscovered concepts such as multiplication, natural numbers, prime numbers and the unique factorization theorem.

System	Focus	Author(s)
AM	Model-driven heuristic evaluation	Lenat
COPER	Knowledge-based generator	Kokar
ABACUS	Qualitative & quantitative, model & data-driven	Falkenheimer, Michalski
BACON, GLAUBER, DALTON, STAHL	Numerical laws from raw data	Langley, Simon, Bradshaw
FAHRENHEIT	Numerical laws scope and experiment generation	Zytkow
IDS	Numerical & qualitative laws, time variance	Langley, Nordhausen
KEKADA	Experimentation strategy	Kulkarni, Simon
GENSIM	Qualitative theory modification	Karp, Friedland
GORDIUS	Theory formation and debugging	Simmons
MECHEM/STOICH	Simplest theory formation	Valdés-Pérez
CER	Comprehensive model of scientific research	Kocabas

Table 2: A Review of Scientific Discovery Research in AI.

AM represents its concepts as frames. The heuristic rules contained in the system are attached to slots in the concepts. The system operates from an agenda of tasks. It selects the most interesting tasks as determined by a set of over 50 heuristics. It then performs all heuristics it can find which should help in executing it. Heuristics represented as operators are used to generalize, to specialize or to combine elementary concepts or relations to make more complex ones. Heuristics can fill in concept slots, check the

content of slots, create new concepts, modify the task agenda or interestingness levels, etc. Because it selects the most interesting task to perform at all times, AM is performing best-first search in a space of mathematical concepts. However, its numerous heuristics (over 200) guide its search very effectively, limiting the number of concepts it creates and improving their mathematical quality.

This system was much more exploratory than mine. It is not trying to explain a data set, but rather explore a space rich in interesting concepts. However, its use of heuristic-driven best-first search applied to frames representing concepts has proven to be powerful. In my system, I represent concepts as frames and perform best-first search in the space of models with the help of domain specific heuristics.

This description and the following remarks concerning MARIKA reflect the understanding of AM's behavior as stated in [Lenat 1977]. AM's disconcerting success seemed to be limited by its impossibility to discover new heuristics. EURISKO was Lenat's attempt at addressing this lack. The difficulties arising during the implementation of EURISKO as well as the controversy surrounding AM lead [Lenat and Brown 1984] to revisit the system and provide interesting conclusions regarding knowledge representation for discovery system.

> The central argument here is the following:
> (1) 'Theories' deal with the meaning, the content of a body of concepts, whereas 'theory formation' is of necessity limited to working on form, on the structures that represent those concepts in some scheme.
> (2) This makes the mapping between form and content quite important to the success of a theory formation effort (be it humans or machines).
> ... [Lenat and Brown 1984]

Though MARIKA doesn't find the model patches on its own, I believe it achieves the goal of mapping form to content in a natural manner. The model is represented as a set of equations objects with left hand-side and right hand-side expression objects that can be

either simple signals, or combinations of signals and/or variables. Similarly, signals comprise a set of slots including shapes slots (constant, linear, exponential, etc.) that are represent as structured objects themselves[2]. The mapping between the physiological defects and the equation patches is direct and natural. Modifying a parameter limit or an expression in an equation (affecting the form of the model) translates directly into a subject out of the range of normals or a recognizable vestibular pathology (interpretation of the content).

II.2.3. BACON

The BACON series of systems were developed over a number of years. However, they are all improvements on the theme of heuristic search for numerical laws. The system is given a set of independent terms and requests the corresponding values of the dependent terms. The system produces higher level terms and intrinsic properties[3] and can apply the same technique over again until it discovers numerical laws relating them. These laws express linear relationships only, but the creation of theoretical terms allows it to discover laws such as Kepler's $\frac{d^3}{p^2} = k$. Similarly, the creation of intrinsic properties allows BACON to postulate numeric terms associated with observable ones. For example, on its path to discovering Black's law of heat, the system postulates the existence of specific heat and finds appropriate values for the substances involved in its experimentation. According to its authors, BACON's weakest point is its inability to establish the limits of validity of the laws it discovers. The FAHRENHEIT system addresses that issue.

---

[2]Knowledge representation is detailed in section IV.4.

[3]Intrinsic properties, such as specific heat, are numeric terms which are postulated by the scientist (or the discovery program) that are associated with the observable nominal ones.

BACON represents data in clusters. These clusters are attribute-value pairs representing observations at any level of complexity. Heuristics are represented as productions rules arranged in four categories: data gathering, regularity detection, higher-level terms definition and calculation, introduction and manipulation of intrinsic properties.

BACON can be viewed as a necessary upstream step for my system. Numerical laws have to be discovered but also explained from simpler principles to provide the knowledge embedded in MARIKA's model and patches. Also, a system trying to model the adaptation of the vestibular system to space could require the addition of signals, and parameters that could benefit from the techniques developed for the introduction of intrinsic properties. However, the authors of BACON say:

> "A hypothesis that will be central to our inquiry is that mechanisms of scientific discovery are not peculiar to that activity but can be subsumed as special cases of the general mechanisms of problem solving." [Langley et al. 1987]

I believe that this is little more true than stating that GPS [Newell and Simon 1963] is the technique that will allow computers to perform the problem solving tasks humans encounter. This theory of scientific discovery seems far too general and has to be specialized and proven at a finer level of detail before such generalization can be justified. However, the insight that scientific discovery is amenable to computational techniques remains at the core of this work.

II.2.4. IDS

IDS formulates both qualitative and quantitative laws. It interacts with a simulated world from which it gathers data represented as attribute-value pairs that are associated with

specific objects and vary over time. Qualitative schemas are used that comprise a finite state diagram. Each state correspond to an interval of time during which the direction of variation of all attributes doesn't change. This representation is directly inspired by Forbus' Qualitative Process Theory [Forbus 1984]. Initially, IDS contains only a simple set of qualitative schemas. Three operators allow IDS to create new qualitative schemas to represent new data. It can create new schemas if an observation cannot be classified in an existing schema. It can add new transitions between qualitative states if they are observed. It can also specify state descriptions if they are found to be too general.

IDS can also generate numeric laws that are indexed by the objects involved as well as the state in which they hold. The qualitative schemas therefore place numeric laws within a qualitatively defined range of validity and also constrain the search for numeric laws. IDS uses data-driven heuristic search for both constant and linear relations between variables. It also has the capacity to create intrinsic properties that are attached directly to specific objects. The system also discovers that some state transitions occur whenever the a particular term reaches a certain value. The authors [Langley and Zytkow 1989] claim:

> "IDS's greatest significance lies in its attempt to integrate the discovery of qualitative and quantitative laws."

I have retained the use of a qualitative description of time varying signals to help constrain the search for a better model. The search spaces however are very different, but the discoveries of my system incorporate quantitative constraints on qualitatively modified model.

II.2.5. KEKADA

The emphasis in KEKADA is placed on discovering strategies and processes that will allow a discovery system to gather the data it requires [Kulkarni 1987, Kulkarni and Simon 1988]. BACON, for example, makes no attempt to justify where appropriate data is coming from or how it was chosen for presentation to the system. The data is just available for use. It is obvious that the proper choice of data sets can help the discovery system focus on interesting areas.

KEKADA provides experimentation strategies that the author claims are applicable to a large range of scientific domains. The strategies are derived from observation of historical discoveries in the field of chemistry. They include generic strategies such as: focusing on a surprising phenomenon, magnification, application of divide-and-conquer, scoping, factor-analysis, relating to other phenomena, as well as domain specific strategies. KEKADA explores a space of rules containing both hypotheses and strategies and a space of experiments and results. The system is implemented as a production system working on attribute-value pairs used to represent experiments, processes, hypotheses and strategies. The system is surprise-driven. It sets up expectations before an experiment is carried out, and focuses on why those expectations were not met. The system's search is controlled by a set of heuristics implemented as operators such as: experiment-proposers, experimenters, hypothesis-or-strategy-proposers, problem-generators, problem-choosers, expectation-setters, hypothesis-generators, hypothesis-and-confidence-modifiers and decision-makers.

My system implements part of this strategy. MARIKA solves a well defined problem: modify its model of the vestibular system to match the experimental results. This can require tuning, parameter range extension or structural modifications. The simulation part

of the system sets up the expectations. As a result of failure in the constraint propagation process, MARIKA posts various hypotheses and performs a simple search to find an appropriate patch and instantiate it. However, KEKADA has no model of the domain, nor is it trying to build one. It is exploring a region by focusing on surprising results and hopes to discover increasingly interesting phenomena without explaining or modeling them.

II.2.6. GENSIM

GENSIM focuses on the hypothesis formation problem and qualitative scientific reasoning. The scientific domain chosen for this system is molecular biology. The knowledge embedded in the system reflects a detailed historical investigation of work done over more than a decade. The hypothesis formation problem occurs when predictions and observations are in conflict. The techniques used in GENSIM assume that the scientific theory can be represented in a model that can be simulated to produce expectations but that also permits hypothesis formation to rectify the faults of the theory. It is interesting to note that because the experimental conditions of an experiment are often known with some imprecision, these conditions can also be modified to improve the quality of the predictions. My system does not consider that possibility because this would be an additional but non interesting mode of modification. This mode is non interesting in the sense that it has no influence on the model. However, finding which conditions to modify is an interesting problem. GENSIM represents knowledge as a hierarchical taxonomy of the classes of chemical entities, the processes that describe chemical reactions and an actual experiment that is made of objects instantiating some of these classes. GENSIM is actually a qualitative genetics simulator. The qualitative reasoning is inspired by Forbus' representation [Forbus 1984] and production rules. Some

new qualitative representations are actually introduced [Karp and Friedland 1989], such as incomplete knowledge of mathematical relationships implemented as partially instantiated frames.

In his study of scientific literature, [Karp 1989] notes patterns in differences between consecutive scientific knowledge levels indicating the reasoning methods used to derive the new theories from the old ones. Among them, object modification (instantiation, postulation, refinement), process modification and creation, parameter refinement, extension and restriction of domain of applicability, and increase and decrease in belief in a theory. Only some of these are clearly implemented in MARIKA although most would prove useful in a vestibular space adaptation system.

Karp also describes four different modes of scientific exploration: confirmation, theory generation, discrimination, and fact finding. It is interesting to compare this model to the classic three step deductive model of theory generation, prediction, and experimentation. It becomes clear that only a single theory can exist at any given time. Discrimination helps select the most promising theory, while fact finding generates knowledge that helps constraint theory generation when it generates too many theories or none. My system is mostly concerned with confirmation and theory formation.

Karp treats the hypothesis formation problem as a design problem. I certainly agree with this approach and complement it with a diagnosis phase. The hypothesis generator, called Hypgene uses backward reasoning to determine what modifications will eliminate the discrepancy. Karp's modification operators perform simple syntactic operations targeted at specific errors. Because of the highly dynamic and coupled nature of the vestibular domain, such technique isn't applicable in MARIKA. I instead rely on a partially qualitative forward reasoner implemented as a constraint network. Constraint violation are directly indexed in a table of possible patches that are instantiated appropriately. As in Karp's system, this simple scheme provides good results. Both systems would benefit from more versatile operators, but the overall architecture remains valid.

II.2.7. MECHEM/STOICH

The purpose of MECHEM/STOICH is given by [Valdés-Pérez 1990] as:

> "Given observed data about a particular chemical reaction, discover the underlying set of reaction steps from starting materials to products, that is, elucidate the reaction pathway."

The system proceeds in three steps. First, it generates pathway hypotheses based on general chemical knowledge. In the second step, a set of heuristics is applied to design experiments and gather additional non-experimental data relating to these hypotheses. Lastly, the system uses the evidence to confirm or invalidate the hypotheses. The system has to propose the existence of unseen chemical elements and does so on the basis of simplicity.

An important factor in the success of a theory is its degree of "parsimony". That is to say that the more concise the theory, the better. Of two theories otherwise equivalent, the simplest and most elegant will always be preferred. This is a theoretically pleasing concept that is rather difficult to apply in practice. The notion of parsimonious theory is present in my system. Modifying the existing theory incrementally by choosing the simpler patches first will likely lead to adequate theories closely related to the old ones. Once again, if the role of innovative science is to provide deep insights by parting significantly from established scientific ground, the role of routine science is tuning the established theory while extending it at the boundaries. Parsimonious changes are a part of the daily routine of science.

Valdés-Pérez introduces a distinction between two types of architecture for scientific discovery systems. The first type, of which his system is an example, is derived from the generate-and-test paradigm. Of course, the basic mechanism can be extended by

generating the hypotheses in a favorable order and interleaving evaluation and generation tasks. However, the second architecture mimics the reasoning of the scientists more closely by establishing small islands of hypothesis and confirming them before linking these islands into a larger theory. I couldn't agree more with this second method and the focus of my thesis is indeed the weaving of these hypothetical threads of into the theoretical fabric.

The author also reflects on research methods in AI:

> "One way to carry out AI research...is to endeavor to invent or improve a general method, concept, or algorithm, then demonstrate its applicability on several problems extracted largely from the real world.
> ...
> A second way to carry out AI research is by selecting a real-world problem whose automation is not easily perceived, and automating it by whatever means found effectual. By analyzing a sufficient number of successful automations, tentative hypotheses may be formed with a degree of credibility assured by their origin in the empirical data of working programs.
> Conducting significant, empirical research in the second manner involves 1. selecting a task largely derived from the real world, 2. demonstrating that its automation is not easily perceived, 3. exhibiting a program that is successful to a certain degree, and 4. analyzing how the program works and the relation of its mechanisms with those of other programs."

I have clearly chosen the second method and this thesis is written with this agenda in mind. However, I strongly disagree with Valdés-Pérez's conclusion referring to his work and stating that:

> "Notably lacking from this list is the proposal of a theory or hypothesis; we believe that not every new datum should give rise to a new hypothesis."

A thesis comprises more than a single new datum. In any case, this argument is only valid at the infancy of a domain when obtaining data points is the primary objective before establishing qualitative laws. I firmly believe that automated scientific formation has reached beyond that stage and any piece of scientific research in this domain should begin by the formation of a hypothesis. The purpose of the rest of the work is then to try to invalidate the hypothesis. Failing this, the hypothesis is often considered valid and, as time passes, becomes part of the theoretical foundation.

MECHEM/STOICH provides a mechanism for conjecturing hidden chemical entities. Though this task is not addressed in my thesis, it would be interesting to transpose it to hidden signals or processes of the vestibular system. Finally, this systems proves that constrained search in order of parsimony is a useful technique. My thesis applies a similar technique to another current scientific domain.

II.2.8. CER

CER is more of a framework for automated discovery than a program automating some tasks on the discovery pathway. Its scope encompasses formulating and choosing research goals, choosing strategies, proposing experiments, and generating, testing and modifying hypotheses. It seems unfair to judge such a system on its level of implementation. CER implements a restricted blackboard architecture through messages posted by the various control modules. Four have been implemented: the goal setter, the goal chooser, the strategy proposer and the experiment proposer. Using this restricted set, the program is able to pursue the rediscovery of superconductor materials unknown to it. It is clear that if automated discovery is to succeed, systems will have to be designed that

comprise many specialized subsystems. This observation has been formulated in [Kocabas 1992] as:

> "The design of a comprehensive computational model of discovery must ultimately include all the essential elements of scientific research covering a number of different research tasks..."

The work presented in this thesis represents the automation of one such task. My claim is that this is a non-trivial task to automate but that it plays an important role in routine science. Citing [Kocabas 1992] again:

> "Modern scientific research is a complex enterprise usually requiring a large number of small but necessary inventions and discoveries of tools, techniques and subsidiary hypotheses before its main goals and strategies are accomplished."

II.2.9. Flite

The Flite system [Prager et al 1989] is a flight simulator tuner. The system helps simulation engineers verify and tune aircraft aerodynamic models to provide the pilot in training with more realistic feedback. The flight program is tested by comparing maneuvers and experiments. Maneuvers are performed by the actual aircraft and the results are recorded as time histories. Maneuvers consist of a steady-state motion, followed by varying pilot inputs and aircraft response. During tuning, experiments are repeated until the time histories of the simulator and the aircraft match. The dynamics are modeled by a set of first order linear equations. The constants appearing in the equations are called gains and represent the influence of state variables on one another. Discrepancies are calculated between maneuver and experiment signals. The simulation

engineer can decide to tune the model to reduce any deficiencies or to tune the pilot inputs of the experiment. Certification requires roughly a hundred maneuvers with different combinations of aircraft configurations. Signals are segmented and discretized using an extension of Forbus' theory [Forbus 1987]. The search in the space of modifications of the simulator is guided by the qualitative representation of the discrepancies. The tuning system heuristically chooses the experiments to perform. It tries experiments affecting the fewest parameters first. However, once a parameter is identified as deficient, the system runs all experiments that could help in the diagnosis. The system then discretizes the maneuver and the experiment and calculates discrepancies. Heuristics are used to decide qualitatively which parameters to modify and how to do so after each test. These qualitative suggestions are then merged to form a consistent set. Numerical optimization is then performed according to the qualitative guide.

Many of the ideas of my system were inspired by Flite. The segmentation problem is simpler in the case I consider because the shape of the input defines it. The types of equations necessary to model the vestibular system are somewhat more complex that the ones used in Flite's aerodynamic model. Flite focuses on the tuning process whereas my system uses tuning only to prove that the data is consistent with the theory. In case of contradictory qualitative suggestions, Flite stops short of action:

> "The third possibility is to interpret contradictory constraints as evidence of a *structural* deficiency. Structural deficiencies are not currently addressed in this work." [Prager et al 1989]

I have focused most of my attention on fixing such structural deficiencies. However, the overall framework proved to be useful and the tuning portion of the system can run into interesting normal range exceptions.

II.2.10. GORDIUS

[Simmons 1988] addresses theory formation and debugging in the course of explaining geological strata. Given a description of a section of terrain both before and after geologic alteration, his system builds a possible geological history, or interpretation. This theory of terrain alteration is not arrived at directly. Simmons proposes a Generate, Test, and Debug (GTD) paradigm. The GORDIUS system first Generates an approximate interpretation using a rule-based system. This interpretation is then Tested with a quantitative simulator which represents passage of geological time. If the simulation doesn't coincide with the observed soil layers, a Debugger traces back through the dependency tree built by the simulator to remove assumptions which are responsible for the bug. GORDIUS considers assumptions about the geological events, the parameter bindings, the ordering of events, the persistence of attribute values, the persistence of objects, and the existence of objects.

In order for the debugger to reason backward from the simulation bug, the simulator must insert all possible assumptions at every point of the dependency network. The processes described in GORDIUS are mostly invertible which permits backward reasoning. As was noted in the discussion of GENSIM, the vestibular domain considered for MARIKA doesn't lend itself to such inversions because the all processes occur simultaneously and show cross-coupling effects that cannot be neglected. Also, MARIKA doesn't generate the model it first starts with, it is given. MARIKA performs only a test and a debug phase. The debug phase of MARIKA considers the possible absence of a process or a signal and questions the binding of parameters. These assumptions are very similar to ones considered in GORDIUS. However, there is no clear notion of event (or process) ordering in MARIKA because of the dynamic nature of the vestibular system. Similarly, objects (or signals) persist; at worse they can decay to zero.

II.2.11. Conclusion

Table 3 summarizes the features used in the systems described above and in MARIKA. The table illustrates that each system satisfies its own requirements according to the type of discovery it performs as well as the idiosyncrasies of the domain it is tackling.

Feature / System	Design	Diagnosis	Frames	Rules	Qualitative	Quantitative	New items	Scope	Heuristics	Model
AM			X	X	X[4]				X	
BACON				X		X	X		X	
IDS					X	X	X	X	X	
KEKADA				X		X			X	
GENSIM	X		X	X	X					X
MECHEM STOICH	X					X	X		X	
CER			X	X	X				X	
Flite		X	X	X	X	X		X[5]	X	X
GORDIUS		X	X	X		X			X	X[6]
MARIKA	X	X	X		X	X			X	X

Table 3: Comparison of relevant systems feature sets.

Scientific discovery efforts often belong to one of two classes. The first class envisions an architecture where every aspect of scientific discovery is taken into account in logical but unimplemented ways. Others pretend only to have built a brick to add to the domain's foundation. While I agree with [Langley and Zytkow 1989] that:

---

[4]AM learns concepts in a symbolic fashion though many of them relate to number theory.
[5]Flite scopes its changes within the qualitative segmentation it defined.
[6]GORDIUS has only a weak model of the domain because, for example, the pre-conditions of a geological process are necessary but not sufficient conditions to its execution.

> "One ultimate goal of research in machine discovery is the construction of an integrated discovery system that covers many aspects of scientific reasoning..."

I believe that until it is clear that we have enough bricks it is more productive to make more bricks than it is to start a wall. However, I am concentrating on routine science so I could be biased toward small constructive efforts and away from unifying visions. Consequently, I disagree with conclusions of the sort found in [Kocabas 1992].

> "...integration of various research tasks and activities...constitutes a step towards building more comprehensive models of scientific research."

Many discovery systems in AI use well documented historical discoveries or toy problems. It is clear that the latter may be unconvincing because the small world of a toy problem is too perfect compared to an actual scientific domain. On the other hand, proving that somehow the solution to a historical re-discovery system was not accidentally coded into the system is difficult, because the computer scientist clearly knows the answer *a priori.* Moreover, such re-discovery systems rarely prove their usefulness by being tested in a scientific domain in its current developmental stage. I have chosen to build a novel system that modifies the theory contained in a model of the normal human vestibular system. In the next section I describe the vestibular domain and the modeling approach I used to represent it.

CHAPTER 3

The Vestibular Domain and its Modeling

"Les théories mathématiques possèdent la propriété d' « exister » sous une forme autonome, mais cette qualité fait aussi leur faiblesse. Dans les sciences de la nature, et en particulier en biologie, les contraintes pesant sur la théorie sont beaucoup plus sévères qu'en mathématiques. Celle-ci, certes, doit être irréprochable dans sa cohérence interne et dans sa logique, donc satisfaire le mathématicien. Mais elle doit aussi adhérer étroitement à une réalité extérieure. Une théorie biologique n'a de sens que si elle correspond à une « représentation » d'objets ou de phénomènes naturels et, de ce fait, peut être directement soumise à l'épreuve de l'expérience." [Changeux 1983]

Mathematical theories have the advantage of "existing" on their own. However, this is also one of their drawbacks. On the other hand, theories in the physical sciences, and in biology in particular, have to satisfy more constraints than mathematical theories do. They clearly have to be perfectly coherent internally and thereby satisfy the mathematician. But they must also closely comply with an external reality. Biological theories make sense only if they "represent" natural objects or phenomena and can therefore be subjected to experimental testing.

III. The Scientific Domain

III.1. The Vestibular Domain

I originally chose the adaptation of the vestibular domain to weightless conditions as my scientific field of experiment. The human orientation system is comprised of the brain merging information from multiple sensors yielding a sense of position and velocity in three dimensional space. Current research focuses on how the central nervous system integrates this information to generate appropriate responses. Providing selective stimuli to a single sensor or suppressing a single signal is often impractical or unethical for human subjects. However, the micro-gravity environment provides altered conditions under which the signals due to gravity are completely eliminated. The experiment I originally chose for this investigation is performed both on the ground and in space. Comparison of these two experimental conditions provides insight into the role the gravity receptors play in spatial orientation. My initial goal was to design a system that could predict the behavior of the vestibular system for a set of astronauts in micro-gravity from baseline ground data using a vestibular domain model. Spacelab Life Sciences 1 (SLS-1) presented a perfect opportunity for such a test. This space shuttle mission included a battery of vestibular tests performed pre-flight on the both the orbiter crew and the payload crew [NASA 1991]. A few data points were also obtained in space for two experiments that can be readily compared to simulated data from our model. However, these data are so inconsistent with the current views of the vestibular system that they cannot be predicted by incrementally modifying the model. Because further space data will be difficult to obtain, I have been forced to shift to a more reliable data set from clinical vestibular research. The techniques developed here are readily usable to help solve the original model development task. However additional structural modifiers, such

as the addition of a new parameter or signal, will be necessary to parallel the full reasoning of the scientist.

The human orientation system comprises multiple sensors sending information to the brain, where it is merged to yield a sense of spatial orientation. Sensors of interest to this study include the otolith organs, the semicircular canals, the visual system, tactile receptors and proprioceptive receptors. The otolith organs and semicircular canals are located in the inner ear and form the vestibular system. The otolith organs react to gravito-inertial forces. They indicate tilt with respect to gravity and linear acceleration. During weightlessness the absence of gravity produces novel sensations. The semicircular canals sense head rotation but provide misleading information during low frequency stimulation (< 0.01 Hz) due to their decay characteristics. The visual system is a more widely known sensory system that generates compelling orientation sensations that sometimes may be erroneous. Loss of balance in very wide field of view theaters is a striking example. Tactile perceptions allow us to sense surfaces in contact with our body and provide valuable orientation cues. For example, a good reference point is the feet pressing against the ground, whereby we can estimate the vertical. Proprioceptive information consists of the relative position of body parts. A proprioceptive reflex of interest in this study is the tendency to right the head toward the perceived vertical. The vestibular system is discussed at much greater length in [Young 1984].

III.2. The Human Orientation Model

Earlier human orientation models often used elements representing the vision system and the semicircular canals. Two influential models are the positive feedback model by

Robinson [Robinson 1977] and the feed-forward paths model by Raphan and Cohen [Raphan et al. 1977]. These models demonstrate good compliance with simple orientation experiments such as rotation about an earth-vertical axis. They were later expanded to represent the otolith organs in an ad hoc manner. However, they do not offer much in the way of explaining the findings of, or providing predictions for other experiments.

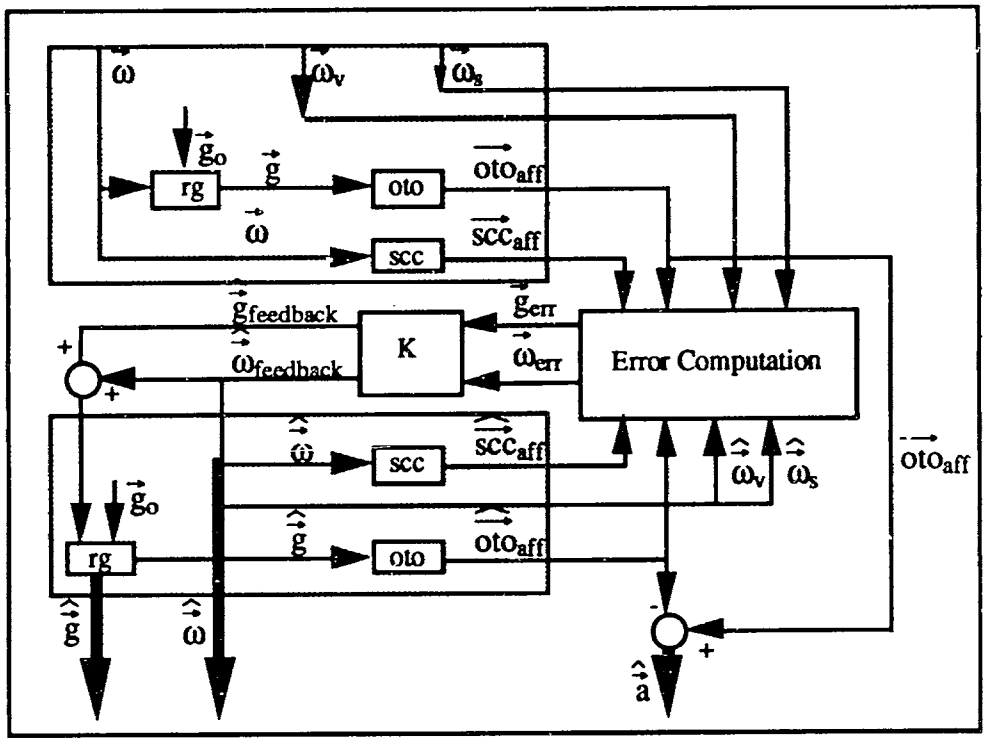


Figure 1: The full human orientation model showing the sensors, the CNS model, the error computation and the feedback calculations

My modeling approach is more systematic and based on the observer theory ([Kalman and Bucy 1961], [Oman 1988]). Simply put, the Observer Theory Model (OTM) states that the system (the Central Nervous System (CNS) in my case) has a model of the body dynamics as well as a model of the sensory dynamics. With this model, the CNS calculates expected values for the sensor signals according to estimates of the body position and the stimulus. It then compares these expected signals to the actual signals being sensed. The difference between the two is then input into the OTM that drives the

feedback loops thereby providing a better CNS estimation. Such an OTM has been developed ([Merfeld 1990], [Merfeld et al. 1992]) for the orientation system of the squirrel monkey that includes otoliths organs and semicircular canals only. I have modified the characteristics of the end-organs to reflect known human reactions. I have also added vision, and tactile and/or proprioceptive signals in order to be able to simulate a wider range of experiments.

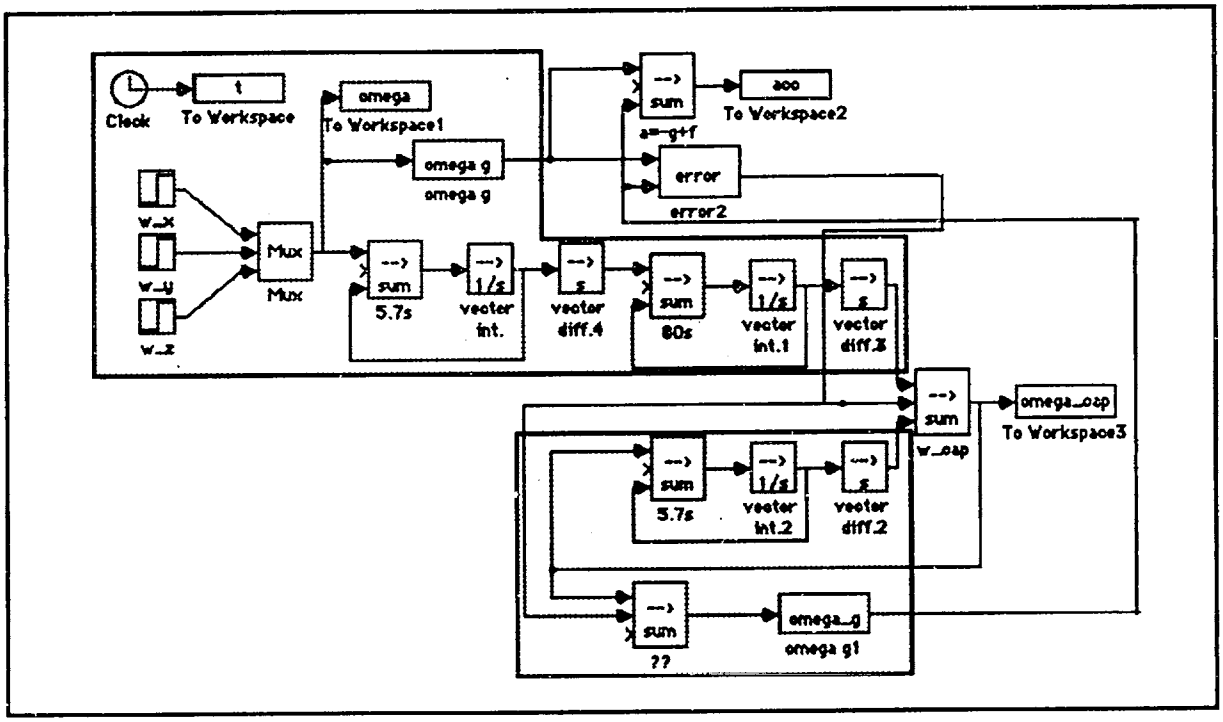


Figure 2: The SIMULAB implementation of the model

The model as shown in Figure 1 highlights the two parts of the computer simulation. The top stippled rectangle comprises the stimuli and the actual sensors, while the bottom stippled rectangle represents the CNS internal model. The central part of Figure 1 is the K matrix that provides the feedback coefficients. I assume that the CNS has perfect knowledge of the sensory dynamics, which means that the sensor sub-models, such as the semicircular canals and the otoliths, are identical to the actual sensors. Also, the subject exerts no motor control, which is an additional simplification.

The model has been implemented in Extend [Copyright Imagine That, Inc. 1990], a computer simulation program for the Macintosh. The model has also been ported to SIMULAB, a modeling extension to MATLAB [Copyright The Math Works, Inc. 1991] as shown in Figure 2. The model accepts motion and visual stimuli and produces three responses: perceived linear motion, perceived angular response, and perceived gravito-inertial vector. For ease of implementation, the model is transposed to the Fourier domain. Therefore, the end organs are represented as combinations of summers, amplifiers, differentiators and integrators. Because of these process arrays have x, y and z components, the model is correct in three dimensional space.

Looking at Figure 1 in some detail, we can observe the following. Gravity $(\vec{g})$ is primarily sensed by the otoliths organs. The gravity error $(\vec{g}_{err})$ is calculated as the rotation required to align the expected measurement of gravitational force $(\widehat{\vec{oto}}_{aff})$ with the actual measurement of gravitational force $(\vec{oto}_{aff})$ sensed by the otolith organs.

Angular velocity $(\vec{\omega})$ is primarily sensed by the semicircular canals (scc), though vision and somatosensory cues have demonstrated their importance particularly during low frequency stimulation. Therefore, the semicircular canal error, the visual error and the somatosensory discrepancy are all calculated and combined in the error rectangle.

The semicircular canal error $(\vec{scc}_{err})$ is defined to be the difference between the actual semicircular canal afference[7] $(\vec{scc}_{aff})$ and the expected semicircular afference $(\widehat{\vec{scc}}_{aff})$:

$$(\vec{scc}_{err}) = (\vec{scc}_{aff}) - (\widehat{\vec{scc}}_{aff})$$

[7]An afferent nerve carries information as a time varying excitation level. Such a nerve exhibits a resting firing rate which increases when excited and decreases when inhibited.

The visual error is defined to be the difference between the angular velocity of the visual field $(\vec{\omega}_v)$ and the expected angular velocity of the head $(\hat{\vec{\omega}}_v)$. This quantity is calculated in the error rectangle and is referred to as residual visual velocity $(\vec{r}_{vv})$:

$$(\vec{r}_{vv}) = (\vec{\omega}_v) - (\hat{\vec{\omega}}_v)$$

Similarly, the somatosensory discrepancy $(\vec{s}_d)$ is defined to be the difference between the angular velocity sensed through somatosensory stimulation $(\vec{\omega}_s)$ and the expected angular velocity $(\hat{\vec{\omega}}_s)$:

$$(\vec{s}_d) = (\vec{\omega}_s) - (\hat{\vec{\omega}}_s)$$

The angular velocity error $(\vec{\omega}_{err})$ is defined to be the weighted vector sum of each of the sensory errors and is calculated in the error rectangle:

$$(\vec{\omega}_{err}) = w_v^{scc} * (\overrightarrow{scc}_{err}) + w_v^v * (\vec{r}_{vv}) + w_v^s * (\vec{s}_d)$$

where w_v^{scc}, w_v^v, and w_v^s are the weights[8] assigned to the semicircular canals error, residual visual velocity, and somatosensory discrepancy, respectively.

The gravity error and the velocity error are transformed into gravity feedback and velocity feedback by multiplying the vector error by the feedback matrix (K):

[8]These weights are matrices of dimension 1x3 and not necessarily multiple of the identity matrix. For simplification I assume them have identical values for all three dimensions.

$$\begin{pmatrix} \vec{g}_{feedback} \\ \vec{\omega}_{feedback} \end{pmatrix} = (K) . \begin{pmatrix} \vec{g}_{err} \\ \vec{\omega}_{err} \end{pmatrix}$$

To greatly simplify this model I assume the cross effects between axes are negligible. This simplification assumes the various sensors to be positioned exactly in perpendicular planes. While it is known that this is not true, the resulting numerical simplification doesn't limit the validity of the results. This simplification yields a feedback matrix with twelve non-zero elements. To further simplify the model I neglect all response asymmetries[9] (e.g. pitch responses are equivalent to roll responses, etc.). This allows us to fully represent the feedback matrix using just four non-zero parameters[10]:

$$K = \begin{pmatrix} k_{11} & 0 & 0 & k_{12} & 0 & 0 \\ 0 & k_{11} & 0 & 0 & k_{12} & 0 \\ 0 & 0 & k_{11} & 0 & 0 & k_{12} \\ k_{21} & 0 & 0 & k_{22} & 0 & 0 \\ 0 & k_{21} & 0 & 0 & k_{22} & 0 \\ 0 & 0 & k_{21} & 0 & 0 & k_{22} \end{pmatrix}$$

Therefore we have:

$$(\vec{g}_{feedback}) = k_{11} * (I) * (\vec{g}_{err}) + k_{12} * (I) * (\vec{\omega}_{err})$$

$$(\vec{\omega}_{feedback}) = k_{21} * (I) * (\vec{g}_{err}) + k_{22} * (I) * (\vec{\omega}_{err})$$

---

[9]The vestibular clearly presents such asymmetries. In particular pitch and roll responses differ from one another and greatly differ from yaw responses. There are also asymmetries with respect to linear acceleration stimuli: horizontal and vertical responses differ. Asymmetries exist even within a single direction as for up and down vertical linear acceleration.

[10]The K matrix is of dimension 6x6. Three dimensions correspond to the three degrees of freedom associated with angular velocity (k_{22}) and the other three dimensions correspond to the three dimensions of space associated with gravitational forces (k_{11}). This allows angular velocity errors to affect the gravitational estimate (k_{12}) and gravity errors to affect the angular velocity estimate (k_{21}).

These feedback vectors are used to drive the CNS models of the otoliths and the semicircular canals respectively, producing expected afferences such as the following (where $\hat{\tau}_{oto1}$ and $\hat{\tau}_{oto2}$ are otolith time constants, $\hat{\tau}_a$ is the scc adaptation time constant and $\hat{\tau}_l$ is the scc long time constant).

$$\widehat{oto}(s) = \frac{s + \hat{\tau}_{oto1}}{s + \hat{\tau}_{oto2}}$$

$$\widehat{scc}(s) = \frac{\hat{\tau}_a * \hat{\tau}_l * s^2}{(1 + \hat{\tau}_a s) * (1 + \hat{\tau}_l s)}$$

The expected afferences are then compared to the actual afferences to produce the error vectors discussed above.

Signal	Description
$(\vec{\omega})$	velocity input vector
rg	processing element calculating the position of gravity in head coordinates
$(\vec{g}_o)$	sensed gravity vector at the beginning of the experiment
$(\vec{g})$	sensed gravity vector
oto	processing element representing the otolith sensors
$\hat{\tau}_{oto1}$	otolith time constant
$\hat{\tau}_{oto2}$	otolith time constant
$(\vec{oto}_{aff})$	sensed otolith afference vector
$(\vec{g}_{err})$	gravity error vector
(K)	9x9 error feedback matrix
(I)	3x3 identity matrix
$(\hat{\vec{g}}_{feedback})$	gravity feedback vector
$(\hat{\vec{g}})$	estimated gravity vector
$(\widehat{\vec{oto}}_{aff})$	estimated otolith afference vector
$(\hat{\vec{a}})$	estimated acceleration vector
$\hat{\tau}_a$	scc adaptation time constant

$\hat{\tau}_l$	scc long time constant
scc	processing element representing the semicircular canal sensors
$(\overrightarrow{scc}_{aff})$	sensed semicircular canal afference vector
$(\vec{\omega}_{err})$	velocity error vector
$(\overrightarrow{scc}_{err})$	semicircular canal error vector
$(\widehat{\overrightarrow{scc}}_{aff})$	estimated semicircular canal afference vector
$(\vec{r}_{vv})$	residual visual velocity vector
$(\vec{\omega}_v)$	sensed visual velocity vector
$(\hat{\vec{\omega}}_v)$	estimated visual velocity vector
$(\vec{s}_d)$	somatosensory discrepancy vector
$(\vec{\omega}_s)$	sensed somatosensory velocity vector
$(\hat{\vec{\omega}}_s)$	estimated somatosensory velocity vector
w_v^{scc}	semicircular canal velocity weight factor
w_v^v	visual velocity weight factor
w_v^s	somatosensory velocity weight factor
$(\hat{\vec{\omega}}_{feedback})$	velocity feedback vector
$(\hat{\vec{\omega}})$	estimated velocity vector

Table 4: Model signals and parameters.

The subtle portions of the model reside in the generation and mixing of the feedback vectors. There are two ways of sensing velocity discrepancy: directly using the velocity sensors (semicircular canals) and using gravity sensors (otoliths) and estimating the rate at which the sensed gravity vector moves away from the expected gravity vector. These estimates are the velocity feedback and the gravity feedback respectively. The semicircular canals detect velocity discrepancies for high frequencies $\left(\geq \frac{1}{\tau_l}\right)$ and the otoliths for low frequencies $\left(\leq \frac{1}{\tau_l}\right)$, with mixing in between.

The parameters and signals of the model are summarized in Table 4. The parameters that are part of transfer functions such as $\hat{\tau}_{oto1}$, $\hat{\tau}_{oto2}$, $\hat{\tau}_{a}$, and $\hat{\tau}_{l}$ represent the response of the end organs. They have a limited range of physiologically possible values. Such afferent signals cannot be investigated in humans directly, but precise ranges of values are known for a variety of animals. Dimensional analysis of the end organs allows us to predict the value of these parameters in the human. Moreover, human orientation psychological experiments measure signals processed by the CNS. These results also constrain the range of value of these parameters. The remaining parameters are less constrained and represent the crux of the human orientation theory presented here. The best possible support for the underlying theory is to set them appropriately for existing data sets and validate predictions with new experiments.

III.3. Conclusion

The human orientation system is a complex system where the brain merges information from a variety of sensors. These signals include the semicircular canals, the otoliths, vision and somatosensory perception. I have designed a model of this system based on the Observer Theory Model. Under this scheme, the Central Nervous System has an internal representation of the sensor organs and tries to minimize the error between its estimate of the sensory nerve afferents and the actual afferent signals. The model is implemented in all three spatial dimensions and provides realistic outputs.

The clinical vestibular domain is well suited for this work because:

- I have a working representation of the model of the system that allows me to run experiment simulations in reasonable amounts of time.

- The model can be represented as a set of differential equations which can be reasoned upon.
- The model is simple enough for someone with a rudimentary understanding of the vestibular system to follow the examples, yet complex enough to make hand analysis prohibitively difficult.
- The model is rooted in current scientific efforts to understand vestibular pathology.
- Data are available to compare model predictions to actual experiment results.
- The domain is sufficiently well understood that interpretation of the results is possible, yet the system could be made useful in a clinical setting.

Moreover, the domain knowledge imbedded in the model could be readily used and expanded in a research effort to solve the original problem of adaptation of the human vestibular system to weightlessness.

CHAPTER 4

Scientific Discovery as a Model-Based Diagnosis and Design Process

At the heart of the natural sciences and engineering there are general notions of **observation** and **measurement**. Based on observation, the scientist builds a physical insight into the problem being studied and from that insight he formulates a theory by trial and error. This theory is a proposed concept of that aspect of nature which he is studying. Guided by this concept he designs new experiments. The observation of the results of these experiments either confirms the theory, dictates a change of the theory or rejects it completely. Although a concept may be beautiful and appealing to the mind of the scientist, the factual results nevertheless are dominant in this interaction of theory and experiment. From this point of view it is legitimate to say that in the fields of natural science and engineering, experiments and observations (measurements) are most fundamental.

Of almost equal importance is the idea of **model building** [Rosenblueth and Wiener 1945]. It can in fact hardly be separated from the observations and experiments mentioned before. The formulation of a theory (as a proposed concept of the aspect of nature being studied) may rightly be called 'model building'; the theory stands as a verbal or mathematical 'model' of reality. [Eyhkoff 1974]

IV. A Model-Based Approach to Scientific Discovery

IV.1. Brief Overview of MARIKA

IV.1.1. Choosing an Appropriate Scientific Domain

The ideal scientific discovery domain should contain some of today's open research problems. Vestibular adaptation to weightlessness is such a problem, but as outlined earlier it presents a number of difficulties. First, there is no guarantee that it is solvable in a reasonable amount of time or solvable at all. Following the SLS-1 results, the principal investigator reflected for a few months before proposing a direction of explanation for the surprising data. If a discovery system made a significant new discovery, it would have to be peer reviewed before it could be accepted. Such delays are prohibitive during testing of a discovery program. To circumvent these difficulties, I chose a clinical domain closely related to the original one. This is not an artificial domain, but one where data are more abundant and better explained. Solutions to the problems I present to MARIKA are known. However, since experts in the domain view the problem as a medical diagnostic problem and I provide solutions in the form of a model of the deficient vestibular system of the patient, I believe my system to be free of hidden knowledge.

IV.1.2. Overview of the MARIKA System

Figure 3 shows a functional diagram of the MARIKA system.

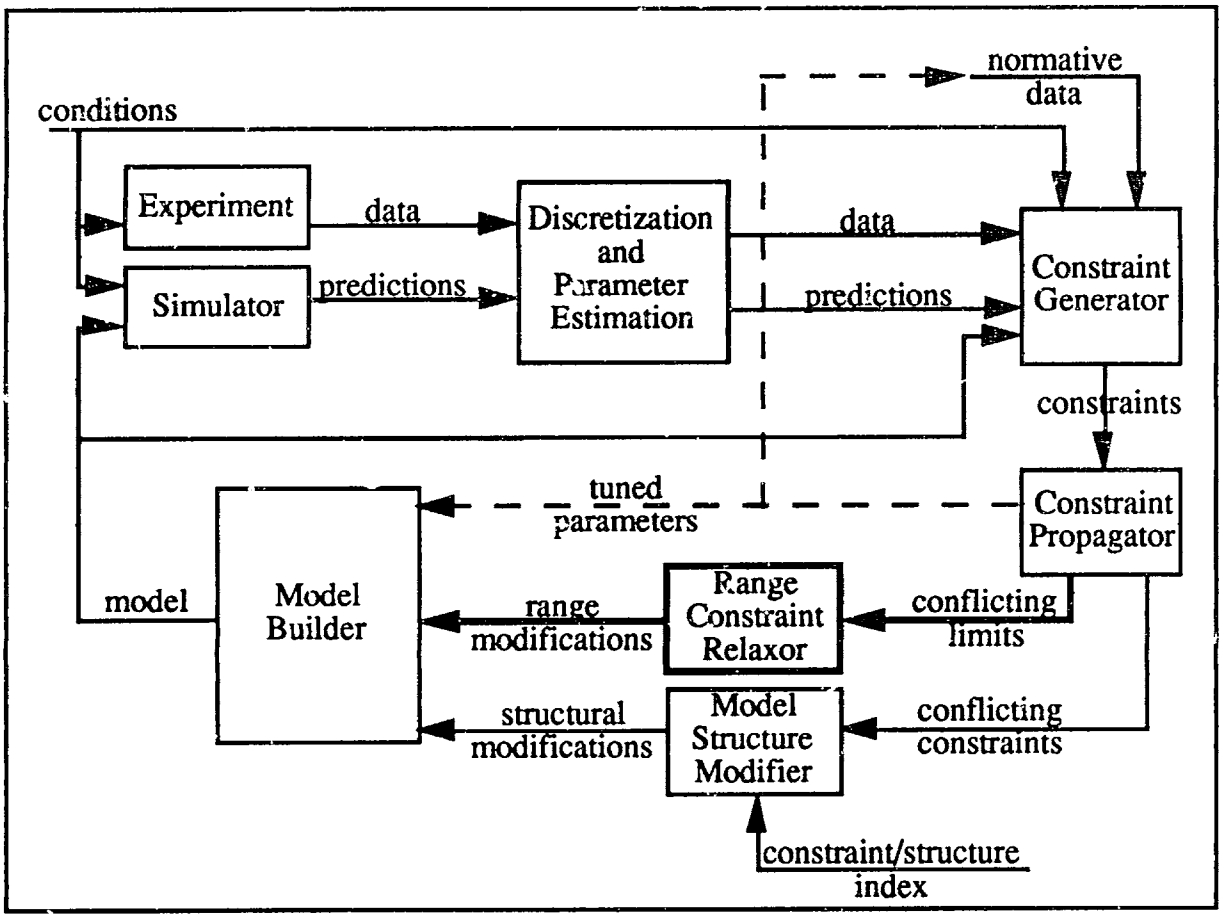


Figure 3: Overview of the system's structure

MARIKA starts with a model of the normal vestibular system. For illustration purposes, in this section I use a simplified model consisting of the semicircular canals as shown in Figure 4. The end organs are represented as high pass filters. The feedback loop implements the exponential decay characteristic of semicircular afference. As in Figure 1, the model clearly displays the stimuli and the actual sensors in the top rectangle, while the bottom rectangle represents the CNS internal model. The central part of the figure is the K matrix that provides the feedback coefficient.

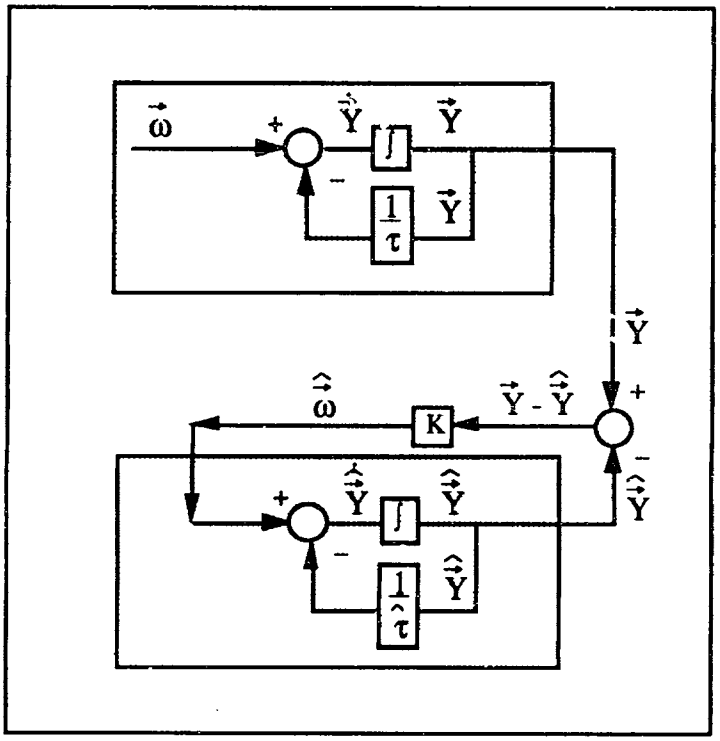


Figure 4: The model of the semicircular canals only

The vestibular model is represented as a set of differential equations[11]. These equations include signals and vestibular parameters linearly combined into expressions. For example, a single dimension of the simple model in Figure 4 can be expressed by the following set of equations:

$$\dot{Y} = \omega - \frac{Y}{\tau}$$

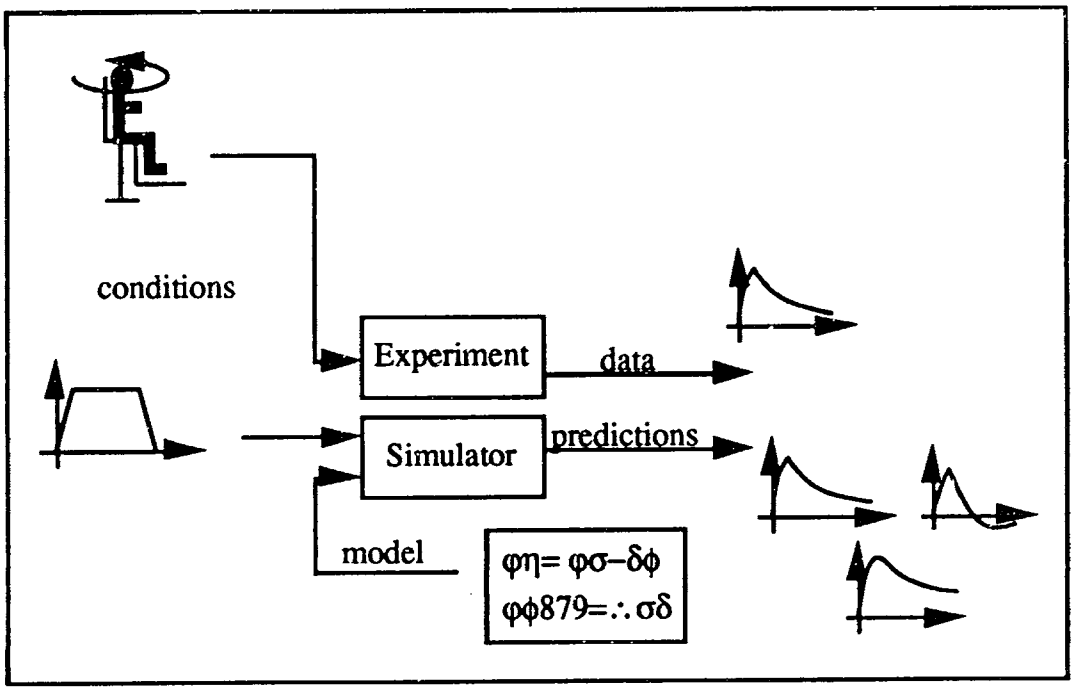
$$\dot{\hat{Y}} = \hat{\omega} - \frac{\hat{Y}}{\hat{\tau}}$$

$$\hat{\omega} = K\left(Y - \hat{Y}\right)$$

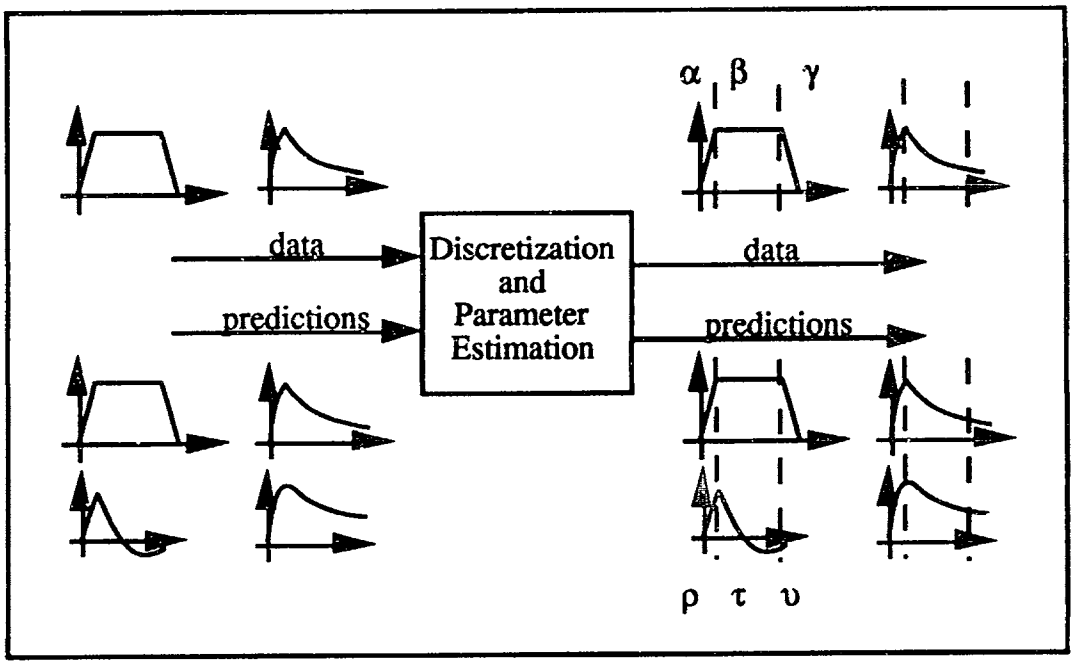
MARIKA works iteratively with a single experiment added at each iteration. Initial conditions are given to the Experiment box and the Simulator box of Figure 3 as

[11]For reason of convenience, the simulation of the model uses a mostly graphic description of these equations as described in [The Math Works, Inc. 1991]. The transformation between the SIMULAB model and the model equations is straight forward. The model is shown in Figure 2.

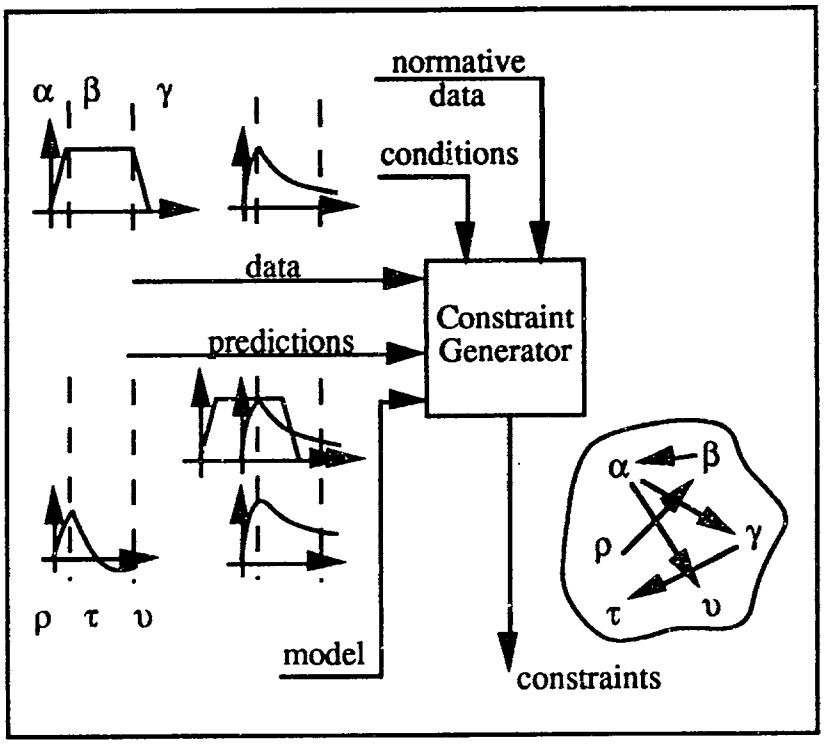
experimental conditions and input signals respectively. For the example of Figure 4, experimental conditions include a rotating chair stimulus with trapezoidal velocity that translates into null visual, and somatosensory input signals, vertical initial gravity conditions and a velocity signal in the vertical axis. The simulator also requires a model that was possibly modified during the previous run of MARIKA. The Experiment produces only output signals, eye movements in this case, while the Simulator produces internal and output signals:



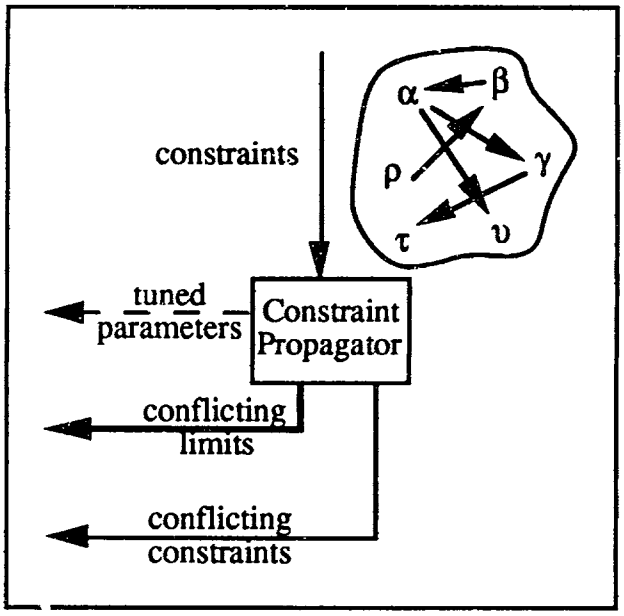
The Discretization box separates the predicted signals into segments and fits them with a set of four basic curves. The Parameter estimation box uses the segmentation and the curves defined by discretization and fit the experimental input and output signals. All the signal parameters are then replaced by variables:



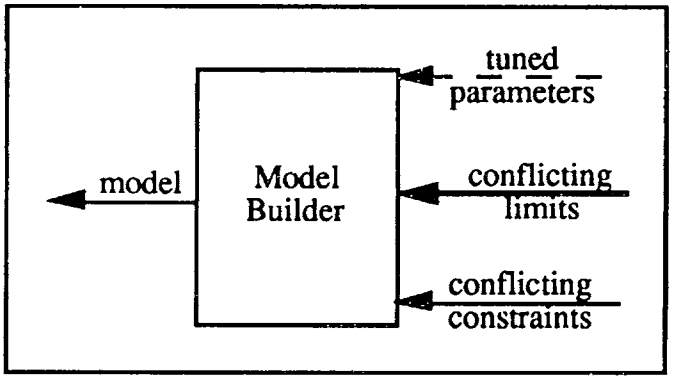
The Constraint Generator uses the discretized experimental and predicted signals, the experimental initial conditions, the normal range of the model parameters, and the model equations to generate constraints on the model parameters and the curve fit variables:



The Constraint Propagator takes the constraint network and tries to propagate all constraints. It either succeeds and generates tighter model parameter limits, or fails on a structural constraint derived from a model equation, or fails on a model parameter limit:



The Range Constraint Relaxer relaxes parameter limits found to be in conflict and outputs a new set of parameter limits. The Model Structure Modifier looks up a table of structural modifications of the model, based on the model equations constraints in conflict, and produces structural modifications. The tuned parameters, the range modifications, and the structural modifications are used to generate a new model. The new model is used during the next iteration of the system.



The MARIKA system can be summarized as executing the following steps for each vestibular experiment:

1) Introduction of the model parameters' range constraints.
2) Simulation of a single experiment with the hypothesized model.
3) Fitting of each vestibular signal generated by the model: internal, inputs and outputs.
4) Generalization of the model signals by replacing the curve fit parameters by variables.
5) Introduction of the constraints derived from initial experimental conditions.
6) Introduction of the constraints derived from differentiation relations between the signals.
7) Introduction of the constraints derived from the model equations.
8) Estimation of the curve fit parameters of the measured experiment data.
9) Propagation of all constraints.

Where step 9) has three possible outcomes:

a) No violation: the system has produced a set of constraints that prove that the individual's vestibular behavior is adequately represented by the current model.
b) Structural constraint violation: the system must look up corresponding structural patches, build the modified models and start the process over.

c) Variable range constraint violation: the system must relax model parameters constraints until all variable range constraints are satisfied.

Using this example, I will illustrate the nine steps described above.

1) Each parameter of the model is represented as a variable having a range of values reflecting the population of normal subjects. For this model, τ, $\hat{\tau}$, and K are represented as:

$$6.0\ s \leq \tau \leq 8.0\ s$$

$$6.0\ s \leq \hat{\tau} \leq 8.0\ s$$

$$1.0 \leq K \leq 3.0$$

which I note as:

$$\tau = [6.0\ ;\ 8.0]$$

$$\hat{\tau} = [6.0\ ;\ 8.0]$$

$$K = [1.0\ ;\ 3.0]$$

2) As the examination of an individual progresses through a series of tests, the system performs simulations of the vestibular apparatus using average parameter values.

3) The predicted data are then discretized to fit the description imposed by MARIKA. The signals are first segmented and then fitted to one of four shapes or simple linear combinations of them: constant, linear, exponential, and sinusoidal. Considering only the signals of the first equation during the first segment we get the following fits:

$$\omega = 0.2 * t$$

$$Y = -9.8 + 1.4 * t + 9.8 * e^{-t/7.0}$$

$$\dot{Y} = 1.4 - 1.4 * e^{-t/7.0}$$

where ω is linear, Y is the sum of an exponential, a linear shape, and a constant shape, and $\dot{Y}$ is the sum of an exponential and a constant shape.

4) The resulting fits are generalized by substituting curve fit variables for the shape parameters :

$$\omega = A * t$$

$$Y = B + C * t + D * e^{-t/\tau_1}$$

$$\dot{Y} = E + F * e^{-t/\tau_2}$$

Three sets of constraints are then introduced on the vestibular variables and the curve fit variables[12]. MARIKA successively introduces initial constraints, differential constraints and model equations constraints.

5) Initial constraints are derived from initial experimental conditions:

$$@\ t = 0 : \omega = 0, Y = 0, \dot{Y} = 0$$

OR

No constraint yielded,

$$B + D = 0,$$

$$E + F = 0,$$

[12]As more experiments are run, the set of curve fit variables is duplicated, but the constraints on the vestibular variables are compounded.

6) Differential constraints enforce derivative relationships between signals:

$$\frac{dY}{dt} = \dot{Y}$$

OR

$$C - \frac{D}{\tau_1} * e^{-t/\tau_1} = E + F * e^{-t/\tau_2}$$

OR

$$C = E,$$
$$\tau_1 = \tau_2,$$
$$\frac{D}{\tau_1} = F$$

7) Model equations are translated into constraints over the model parameters and signals they refer to:

$$E = -\frac{B}{\tau}$$

$$A - \frac{C}{\tau} = 0$$

$$F = -\frac{D}{\tau}$$

8) The clinical data are then segmented and fitted similarly to the simulated data.

$$\text{assuming} : \omega = A * t \,,\, \hat{\omega} = G * \left(1 - e^{-t/\tau_2}\right)$$
$$A = 0.2$$
$$\tau_2 = 21.0$$
$$G = 2.8$$

9) Constraint propagation is then initiated.

a) If the individual is a normal subject, the system constrains the set of parameters describing the vestibular system of that individual in a way similar to [Prager et

al. 1989]. Modifications to model parameters are made within the constraints of normality:

$$\tau = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$K = [1.75 ; 2.33]$$

If the individual is a pathological case, the system is unable to find a consistent set of parameters. The system then modifies the structure of the model according to the discrepancy highlighted during constraint propagation and tunes the model. Discrepancies can be of two types.

b) In case of a structural constraint violation, the system looks up corresponding patches, builds the modified model and starts the process over. Because the model is very simple, the only structural modification consists of removing the feedback loop, which leaves the following equations:

$$\dot{Y} = \omega - \frac{Y}{\tau}$$
$$\hat{\omega} = Y$$

c) In case of vestibular variable range constraint violation, the system must relax model parameter constraints until all variable range constraints are satisfied. For example, if we restrict the range of normals to:

$$\tau = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$K = [1.5 ; 2.5]$$

and observe data such as:

$$\text{assuming} : \omega = A * t \, , \, \hat{\omega} = G * \left(1 - e^{-t/\tau_2}\right)$$

$$A = 0.2$$

$$\tau_2 = 11.0$$

$$G = 1.9$$

MARIKA will return a relaxed set of parameter constraints such as:

$$\tau = [5.0 \, ; \, 6.33]$$

$$\hat{\tau} = [4.47 \, ; 4.80]$$

$$K = [1.5 \, ; \, 1.9]$$

In any case, the final result is a model structurally different from that of a normal subject but that represents the subject's pathological vestibular system. Therefore the theory contained in the model of the individual is modified because that individual is different from the norm, not because the model is wrong for all individuals observed, as would be the case in the original space experiments.

IV.2. Knowledge Representation Details

The model is represented as a set of first-order differential equations. The example in Figure 4 can be expressed as:

$$\dot{\vec{Y}} = \vec{\omega} - \frac{\vec{Y}}{\tau}$$

$$\dot{\hat{\vec{Y}}} = \hat{\vec{\omega}} - \frac{\hat{\vec{Y}}}{\hat{\tau}}$$

$$\hat{\vec{\omega}} = K\left(\vec{Y} - \hat{\vec{Y}}\right)$$

Assuming a simple rotation stimulus around a vertical axis, the only non-zero data are along that axis. I therefore simplify the example further to show only the vertical component. The system of equations simply becomes:

$$\dot{Y} = \omega - \frac{Y}{\tau}$$

$$\dot{\hat{Y}} = \hat{\omega} - \frac{\hat{Y}}{\hat{\tau}}$$

$$\hat{\omega} = K\left(Y - \hat{Y}\right)$$

The constant coefficients or parameters of the model (τ, $\hat{\tau}$, and K in the above equations) are represented as constraints over the range of possible values.

$$\tau = [6.0\,;\,8.0]$$

$$\hat{\tau} = [6.0\,;\,8.0]$$

$$K = [1.0\,;\,3.0]$$

The time varying signals such as input, output, and various intermediary signals are discretized. The only input here is ω, the only output is $\hat{\omega}$, the intermediary signals and their derivatives are Y, $\dot{Y}$, $\hat{Y}$, and $\dot{\hat{Y}}$. The signals are separated into segments with a begin and end time. Those times are governed by that of the input signal which is clearly segmented as shown on Figure 5.

All the segments representing a single signal are put together to represent the discretized signal. Each segment can be approximated by a constant, a ramp, a decaying exponential, a sinusoid or a linear combination of them. A segment is therefore approximated by one or more shapes. These representations can easily be calculated from the raw signals by least-squares algorithms. This signal processing step is illustrated in the discretization and parameter estimation box of Figure 3.

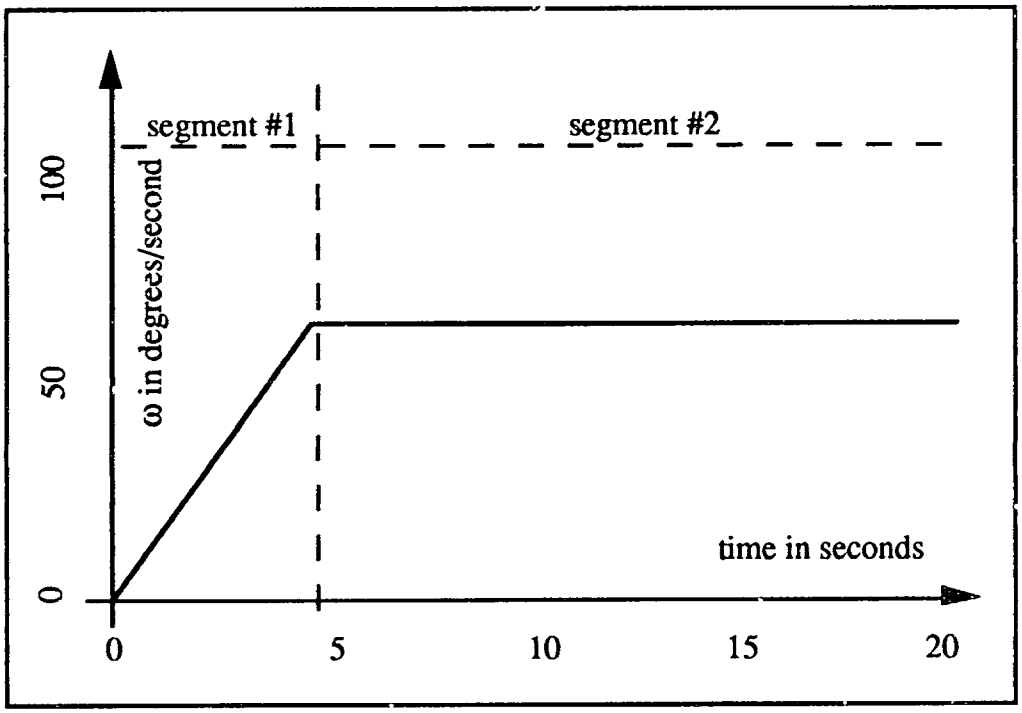


Figure 5: Segmentation of the input signal, ω

This crucial step of data reduction allows MARIKA to transform a continuous data analysis problem into a discrete one. Moreover, because there are only four simple shapes

to consider, the system can apply efficiently four sets of rules and doesn't require an extensive array of mathematical tools. Each signal is reduced to a few segments, which are reduced to a few shapes, which are described by a few parameters. MARIKA can therefore reason at a qualitative level that captures the important mathematical characteristics of the signals over time. Because of the carefully chosen representation, the qualitative reasoning is successful at interpreting data in a numeric, time varying domain.

The limited mathematical reasoner can handle all linear differential equations involving signals that can be represented by combinations of the four shapes. Its operators perform equation reduction, signal comparison, derivation calculation, initial condition verification, and transform all its conclusions into constraints on the model parameters and the shape parameters. Standard constraint propagation is then used to produce the desired relationships between variables.

The reasoner as defined above is complete. Signals are represented as linear combinations of the four shapes and are transformed by linear equations or derivation operators that are linear and each one outputs simple shapes given a simple shape as input.

The vestibular domain as modeled in Chapter 2 is mostly linear. There are three sources of non-linearity, the calculation of the gravity vector $\vec{g}$ from $\vec{g}_o$ and $\vec{\omega}$, the calculation of the estimated gravity vector $\hat{\vec{g}}$ from $\hat{\vec{g}}_o$ and $\hat{\vec{\omega}}$, and the calculation of the gravity error vector $\vec{g}_{err}$ from $\overrightarrow{oto}_{aff}$ and $\widehat{\overrightarrow{oto}}_{aff}$. All three calculations require vector cross products or dot products. These operations are not linear but can be simplified in some cases. When $\vec{\omega}$ and $\vec{g}$ are collinear, $\vec{g}$ remains unchanged, therefore $\hat{\vec{g}}$ is identical to $\vec{g}$ and $\vec{g}_{err}$ is zero. In general, once the system has reached steady state, the calculation of the two gravity vectors requires multiplying a constant velocity component with a sinusoid gravity component. Because one of the terms is a constant, the operation is linear. Also, at steady state, the two gravity vectors rotate around the subject at the same constant speed. They can be represented as sinusoids with identical frequencies but different phases. The

gravity error vector is therefore constant, conserving linearity. The system is still complete for linear differential equations, gravity error calculation and gravity vector calculation under the collinearly and the steady state assumptions.

The representation of the model is largely object oriented and provides a natural mapping between the domain knowledge and the system. Specific representation techniques such as inheritance, deamons, and methods are used to further streamline the system and make for a compact efficient implementation. Details of how this representation is used by MARIKA are given in the following sections.

IV.2.1. Parameter Estimation

Each segment of a signal is approximated by one or more shapes. These representations can easily be calculated from the raw signals by least-squares algorithms.

Segment Type	# of Params	Params Name	Parameter Description	Segment Equation
constant	1	C	constant value	C
ramp	1	A	slope of segment	A * t
exponential	2	K τ	maximum time constant	$Ke^{-t/\tau}$
sinusoid	2	A ω Φ	amplitude frequency phase	$A * \sin(\omega t + \Phi)$

Table 6: Segment representation.

Table 6 details the representation used for each type of shape. The curve fitting algorithms are directly inspired from [Balkwill 1992] and were adapted with help from Ted Liefeld of the Man-Vehicle Laboratory at MIT. As explained in Section III.2, SIMULAB is embedded within the MATLAB numeric computation software package. The core of the algorithm relies on a constrained optimization function with Nelder-Meade search algorithm of the parameters. The MATLAB code is detailed in Appendix A.

Once the segment has been fitted, the precision of the resulting fit is evaluated and stored. Each fitting parameter is then replaced by a unique variable, even if another parameter exists with a numerically close value. The experiment data are fitted in a similar fashion. However, the shapes of each segment are assumed to be identical to those of the simulated data. Parameter estimation [Eykhoff 1974] is then used to cast the experimental data within the model's framework.

IV.2.2. Initial Value Constraints

The initial value of a segment is simply obtained by adding the zero value of all its shapes. The initial conditions of the four shapes are as follows:

constant shape	`constant`
linear shape	0
exponential shape	`maximum`
sinusoid shape	`amplitude * sin (phase)`

Initial constraints are generated by forcing the calculated initial value of a segment to be equal to the imposed value. The imposed value comes from the experimental setup if the

segment is the first of the list representing a signal. If the segment isn't the first, the final condition of the previous segment is the initial condition of the next segment by argument of continuity. The final condition can be obtained from the simulation, from the data or from the discretized signals. It is most efficient to obtain it from the information stored in the segment itself by calculating the value of the shape at the time of the end of the segment.

IV.2.3. Differentiation Constraints

In order to generate derivative constraints, I compare the calculated differential segment to the actual differential segment. The calculated derivative is obtained by creating a segment that has shapes recursively calculated as the derivatives of the shapes of the original segment. The comparison of constant and linear shapes is trivial. On the other hand, the comparison of exponential shapes or sinusoid shapes is more complex. In order for these to be compared, one must decide whether they are compatible based on time constants and frequency respectively. In case of compatibility, constraints are introduced on the characteristic parameters of the shapes to force them equal. If a shape isn't compatible with any other, its maximum or amplitude must be zero respectively.

IV.2.4. Interpretation of the Model Equations

Model equations are viewed as having a left-hand side and a right-hand side that contain expressions. An expression includes an operation (one of +, -, *, and /) and two arguments. If the operation is an addition or a subtraction, both arguments have to be signals or expressions. Otherwise, the first is a signal and the second is a variable.

Using this representation, the first equation of the model shown in Figure 4, $\dot{Y} = \omega - \frac{Y}{\tau}$, can be represented by the network in Figure 6.

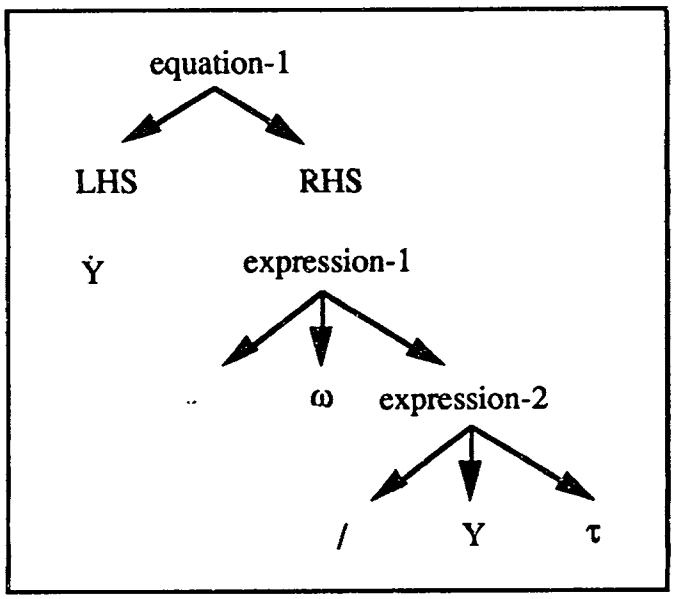


Figure 6: Object network for equation $\dot{Y} = \omega - \frac{Y}{\tau}$

Equations hold at all times, but are interpreted segment by segment. Equations are defined in terms of signals. Each equation is interpreted segment by segment. The algorithm is as follows:

```
interpret equation
     reduce LHS
     reduce RHS
     compare LHS RHS
```

The first task is to reduce both expressions to segments. The definition of `reduce` is recursive. There is no work required if the expression is already a segment. If the expression is an addition or a subtraction, the two expressions are reduced to segments and added together to form a single segment. If it is a multiplication or a division, all the shapes are modified appropriately and gathered to form the new segment. Adding two segments is done by adding similar shapes together. Adding or subtracting constant or

linear shapes requires adding or subtracting their constants or slopes respectively. Exponentials and sinusoids have to be compatible to be added or subtracted. If they aren't, they are just collected in the resulting segment with proper sign adjustment to the maximum and the amplitude respectively if the operation is a subtraction.

To accommodate the non-linearities described earlier in this section, I have defined two other operators called `error` and `delta-g`. They take two pairs of arguments as inputs and output a single shape. Both operators are limited to the collinearity case of the rotating chair and the steady state case. No other case is recognized, and the operators return an error if presented with improper input. The `error` operator produces a constant shape. That shape is zero in the case of collinearity and easily calculated from the phase shift of the inputs for the steady state case. The delta-g operator produces a zero constant shape in the case of collinearity. For the steady state case, two of the inputs have to be constant and the output is a linear combination of the two other inputs. The vestibular system is such that the velocity arguments are constant and the gravity arguments are compatible sinusoids. The output is consequently another compatible sinusoid.

The second task is to compare the segments resulting from the reduction of the expressions. Here again, comparisons are done separately for each type of shape. If the same shapes exist on both sides, their characteristics are asserted to be identical. If only one exists, it is asserted to be zero. Compatibility of exponentials and sinusoids has to be checked before zeroing a shape or asserting two shapes to be equal.

IV.2.5. Constraint Propagation and Patching

IV.2.5.1. Model Tuning

During an experiment, only the input and output signals are known. Indeed, the intermediary signals are located (or hypothesized to be) within the vestibular end-organs and the brain. For obvious ethical reasons, we may not access these intermediary signals. However, every part of the model is accessible and I assume that the intermediary experimental signals are identical to the intermediary predicted signals. Because the input signals to both are identical, only the output signals differ. Once the model data are discretized and the constraint network is set up, the experimental data are discretized and injected into the network. This process is repeated for each test performed on the individual.

Constraint propagation is aimed at resolving apparent contradictions between the constraints posted by various experiments, or between the model structure and the data, or between the experimental data and normative data. An important feature of the constraint network is that it is identical for all normal subjects. If such constraints can be resolved, the resulting set of constraints satisfies the experiment data and predicted outputs. Given additional hypotheses (introduced as constraints) each parameter can be assigned a unique value representing the individual as accurately as possible within the structure of the model. This data path is represented by the dashed arrow of Figure 3. Numerical optimization is not used exclusively because it doesn't capture all the information present in the constraint network and requires additional information. Also, the more numerous the parameters and tests used for tuning, the weaker the dependence of the tuning on each test. The constraint propagation technique uses all the information available and only the information available. Selecting the proper set of parameters to tune is a knowledge intensive task and benefits from the techniques described above.

IV.2.5.2. Structural Modifications

If the constraint sets cannot be resolved, the constraint propagation mechanism terminates on a contradiction. The constraints responsible for the conflict are then used to hypothesize about structural modifications to the system that can most likely predict the data correctly. The possible violations of these constraints are known *a priori* because they reflect possible physiological failures. These should not be confused with possible diagnoses or indices of disorders and symptoms as in [Mira et al. 1988]. Two types of conflicts can arise, variable range conflict and structural conflicts. The first type occurs if the model is structurally valid, but the individual being tested lies outside the range of normals. It is clear that regardless of the distribution of values for a model parameter such as a time constant, a portion of the population is left out. In such cases, the constraint propagation system fails when attempting to narrow one of the limits of a variable. The system must then attempt to relax constraints on the model parameters. This data path is represented by the bold arrow of Figure 3 and is described in Section IV.2.4.

The second type of conflict involves multiple experiments. In such cases, the propagation system fails on a constraint derived directly from a model equation. As described above, the system has to modify the structure of the model[13] and start the process over. This data path is illustrated by the solid arrows at the bottom of Figure 3. When MARIKA fails on a structural constraint, the system determines which constraint was violated and looks up a table of patches. The patch table is indexed by model equations from which model structure constraints are derived directly as explained in Section IV.2.5.2. The patch table

[13]In keeping with my initial goal of scientific discovery, these hypotheses would be methods of revision of the theory as put forward by the scientist before performing the actual experiments. It is clear that in reality such heuristics are often developed a posteriori; however an autonomous discovery system would require a set of such methods. A system for autonomous discovery such as the one described here requires the scientist to perform a priori analysis of the favored theory to identify possible failures of the theory as well as patches. It seems that the pattern of failure itself could direct reasoning and therefore general patching methods have to be complemented by specific failure dependant recommendations.

is displayed in Table 7. The offers only three different types of patches. However, the mechanism could be expanded with additional patches that would enrich the structural range of the system.

Constraint propagation failure	Proposed patches
w1 (single parameter)	Left scc only OR Right scc only
w2 (single parameter)	Left scc only OR Right scc only
any part of equation $\dot{Y} = w_1 * \dot{Y}_R + w_2 * \dot{Y}_L$	Left scc only OR Right scc only
τ (single parameter)	No velocity storage
$\hat{\tau}$ (single parameter)	No velocity storage
k_1	No velocity storage
k_2	No otoliths

Table 7: Structural patches look up table.

The hypothesized models are then tuned in turn as explained above independently from one another. It is interesting to note that even the simplest example leads to two different explanations that differ only in parsimony[14] (detailed example in Chapter 5). The system terminates when all sub-models have been successfully tuned. The most parsimonious model is then chosen. Because all of them have been adequately tuned, criteria such as the number of constraints violated, and therefore number of structural modifications, are used in selecting the favored model. Other schemes in which the structural modifications are ordered by preference or likelihood would bias the system towards those solutions

[14]I measure parsimony by the number of changes a modified model requires from the original model.

and could terminate search on the first successfully modified model. A best-first search would then seem appropriate.

IV.2.5.3. Model Parameter Range Modifications

Constraint propagation is implemented in three blocks: the discretized vestibular data parameter constraint, the constraints derived from the model equations, and finally, the model parameters range constraints. When more than one experiment is considered, the vestibular data and the model equations exist in as many sets as experiments. The blocks are input consecutively. The model parameters still form the last constraint block as sketched in Figure 7.

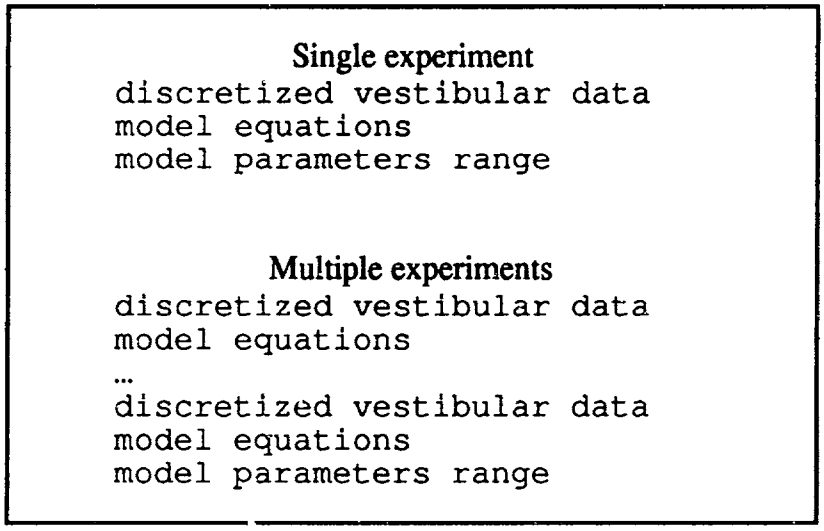


Figure 7: The constraint blocks ordering

When failing on a parameter range constraint, the network gives a message indicating the parameter limit being asserted. As this limit is incompatible with the rest of the network, it is temporarily discarded and put in a queue. Once all offending limits have been discarded, they are re-introduced one by one. They are of course no more compatible than

during the first pass, but additional constraints have been incorporated and therefore MARIKA has more information to base its decisions on.

When reinserted a limit can prove to be binding either directly or indirectly. Inspecting the partial constraint propagation performed by the system, the parameter range restriction can either succeed, but as a consequence some other constraint is violated (indirect path), or the parameter just cannot be limited (direct path). The direct path is the simplest to one to solve. The limit has to be relaxed until it can be satisfied. During relaxation, more general limits have to be enforced, such as a parameter taking positive values only. In the indirect case, the parameter is limited but other constraints have to be relaxed. Any range limit which is still valid is a candidate for relaxation. Valid limits are those which haven't been made any tighter during propagation. Limit relaxation is iterative. Parameters must have absolute limits which they cannot violate as well as normal limits. The first relaxation attempt consists of using the absolute limit instead of the normal one. If successful, the limit can be made tighter by interval dichotomy or any other method. A relaxation is judged successful if the system made some progress by constraining some parameter further, except for the parameter limit being relaxed. If the first relaxation attempt is unsuccessful, another limit is considered. Once a limit has been successfully relaxed, the next range constraint from the queue is re-introduced and the same process takes place until the queue is empty. A detailed example is described in Chapter 5.

IV.2.6. Implementation

The system is implemented on a Macintosh computer using the general purpose programming environment, Macintosh Common Lisp [Apple 1991]. Specialized software packages such as Extend [Imagine That 1990] and SIMULAB [The Math Works 1991]

are used to run simulations and MATLAB [The Math Works 1989] is used to perform the parameter estimation. The Screamer package [Siskind 1991] is used to implement the constraint propagation scheme. Screamer is a macro package running in Common Lisp that I have modified slightly to run under Macintosh Common Lisp. I have made other small modifications to allow documentation of the constraint propagation failure for MARIKA's structural modification and parameter range modification. Modified code is presented in Appendix B.

The signals, segments and shapes are represented as frames. I used the Parmenides 1.5 [Shell and Carbonell 1990] frame system with some custom additions facilitating object-oriented style programming documented in Appendix C.

The programming emphasis is on presenting a complete system in working order. The idiosyncrasies of the Macintosh operating system have not been explored to provide an integrated system. The goal of this work is to demonstrate a conceptual architecture, not a seamless set of interacting modules. The current implementation includes the simulation, the curve fitting, the constraint generation, and the constraint propagation portions of the system. Currently, the operator has to implement the proposed structural patches in the equations and the simulator, has to trigger the modules in the appropriate sequence, and has to follow instructions in relaxing parameter range limits.

IV.3. Model-Based Scientific Discovery as Diagnosis and Design

I consider model-based discovery to be a diagnosis and design problem. More precisely, I see model-based theory refinement as a four step process. First gather data, second compare the data to model-based predictions, third identify the sources of discrepancies

between the predictions and the field data, and fourth fix those discrepancies. The first three steps are traditionally addressed by diagnosis systems while the fourth step requires design techniques. In the following sections I explain both analogies using traditional diagnosis and design terminology and literature to justify them.

IV.3.1. Introduction

As shown in Table 3, only a few discovery systems include a model of the scientific domain they explore. Only GENSIM, Flite, GORDIUS, and MARIKA do. Other systems are heuristic-based and this difference can mostly be explained by the level of discovery they perform. The four model-based systems listed above all assume sufficient development of the scientific domain to allow mathematical modeling. The other systems are often more exploratory or work at one of the law levels defined in Table 1.

I will adopt a functional definition of a model as in:

> "Model building consists of the following steps:
> selection of a model structure based on physical knowledge;
> fitting of parameters to available data (estimation);
> verification and testing of the model (diagnostic check);
> application of the model to its given purpose." [Eyhkoff '974]

In the context of my thesis, the purpose of the model is to concisely represent scientific domain knowledge in a form that can be simulated to produce data predictions. Moreover, the representation of the model and the implementation of the accompanying simulation method must allow symbolic reasoning necessary to debug and improve the model.

IV.3.2. The Diagnosis Analogy

MARIKA starts from a theory expressed in a mathematical model of differential equations. It therefore doesn't consider the theory generation problem for unorganized data. It is however rare for a theory to survive without modifications. I claim that the process of scrutinizing a theory to correct it can be viewed as a diagnosis problem. I will take [Hamscher and Davis 1987] definition of diagnosis:

> A useful way to decompose this [the diagnosis] task is to consider three separate tasks: (i) generating fault hypotheses, (ii) checking those hypotheses for consistency, and (iii) discriminating among the consistent hypotheses on the basis of further probes or tests.

MARIKA implements the first step through constraint propagation. The propagation failure directly indexes possible model modifications called patches. Consistency of the patches is performed through another constraint propagation pass. Discrimination among competing patches requires further debugging and parsimony considerations. In this perspective, there is constant interaction of prediction (data obtained from model simulation) and observation (field data). Davis describes the goal of diagnosis as "Given some observations of a misbehaving device, a description of its internal structure, and descriptions of the behavior of its components, we wish to find out which components[15] could have failed in such a way as to explain the misbehavior". Similarly, I describe the goal of model-based scientific discovery as "Given some valid experimental data, a description of the internal structure of the theoretical model, and descriptions of the behavior of its components, we wish to find which components of the model failed in such a way as to explain all discrepancies and suggest how to fix them". I view model-

[15]Diagnosis of multiple simultaneous faults can be more complicated as the choice of the proper fault cannot rely on explaining all symptoms but most symptoms, and overlapping of symptoms between faults makes discriminating among diagnoses less efficient.

based theory formation as a reversed form of diagnosis where the data are correct but the model is wrong.

A recent paper [Hamscher 1991], describes current issues in diagnosis research. Two of these issues are the capacity of reasoning with defeasible assumptions and interleaving of deliberation and action.

Theory formation involves hypothesis formation. However, the assumptions used are not always correct and may have to be revised. Defeasible assumptions are therefore the crux of theory debugging. However, one assumption that diagnosis systems rarely question is the adequacy of the system description. At best, diagnosis systems manipulate hierarchical levels of representation, whereas the goal of scientific discovery systems is to find how to modify the description to fit the data better.

What I propose for model-based scientific discovery is a form of therapy:

> *Therapy* can be defined as an interleaved process of using both diagnosis and repair to suppress undesired symptoms [Hamscher 1991].

MARIKA takes experimental data sets one by one and modifies its model incrementally. Similarly, a vestibular adaptation theorist program would perfect its theory during flight as individual astronaut data are gathered and dictate changes.

The two main paradigms used in diagnosis systems are probabilistic and logical. The quantity of knowledge regarding space vestibular adaptation that is available to or analyzed by the scientific community is not sufficient to allow true probabilistic considerations. It would seem that scientists reason more in terms of causal relationships than they reuse theory patches which have worked before. However, MARIKA's domain largely overlaps with vestibular clinical diagnosis (see Section III.3) and its selection of theory patches has a probabilistic flavor because it bases its choice of a patch on prior

success. Statistics are not used, but patches are selected only if they proved useful in the past. MARIKA chooses from a look up table. The assumption is that the scientist believes that the patches can be indexed by the type of model failure observed. These patches could be ordered by degree of occurrence in the patients, but this would assume *a priori* knowledge which is not modeled because it doesn't exist in the original domain of vestibular adaptation to weightlessness. For this task, the scientist must provide theory patches ahead of time and statistical data just don't exist. Nevertheless, the scientist can guess which modifications can prove useful.

MARIKA is therefore exploring a space of possible models that can be outlined at the time the lookup table is designed. A more exploratory system would generate model patches and therefore models in a more data-driven dynamic fashion.

The diagnosis approach has to be complemented because while diagnosis is content with locating the fault, assuming either that the component can be repaired or replaced according to established procedures, scientific discovery must design a new model that leaves no unexplained data. Model-based scientific discovery must include a design stage that satisfies the requirements set up by the discrepancy.

IV.3.3. The Design Analogy

Of the systems described in Chapter II, GENSIM is the one that most emphasizes scientific discovery as a design process. More precisely, Karp presents hypothesis formation as design. GORDIUS also includes theory design as its generation stage. However, it is implemented as a rule-based system and Simmons doesn't consider design as an important issue. I believe Karp's idea can be applied more broadly to the entire scientific discovery domain. In a way parallel to engineering design, scientific discovery can range from innovative to routine.

Figure 8 is a summary of the nature of design, design knowledge, and alternative models of design as described in [Sriram and Tong 1991]. Thick arrows specify design concerns applicable to scientific discovery in general. Thin arrows specify design methods used in MARIKA.

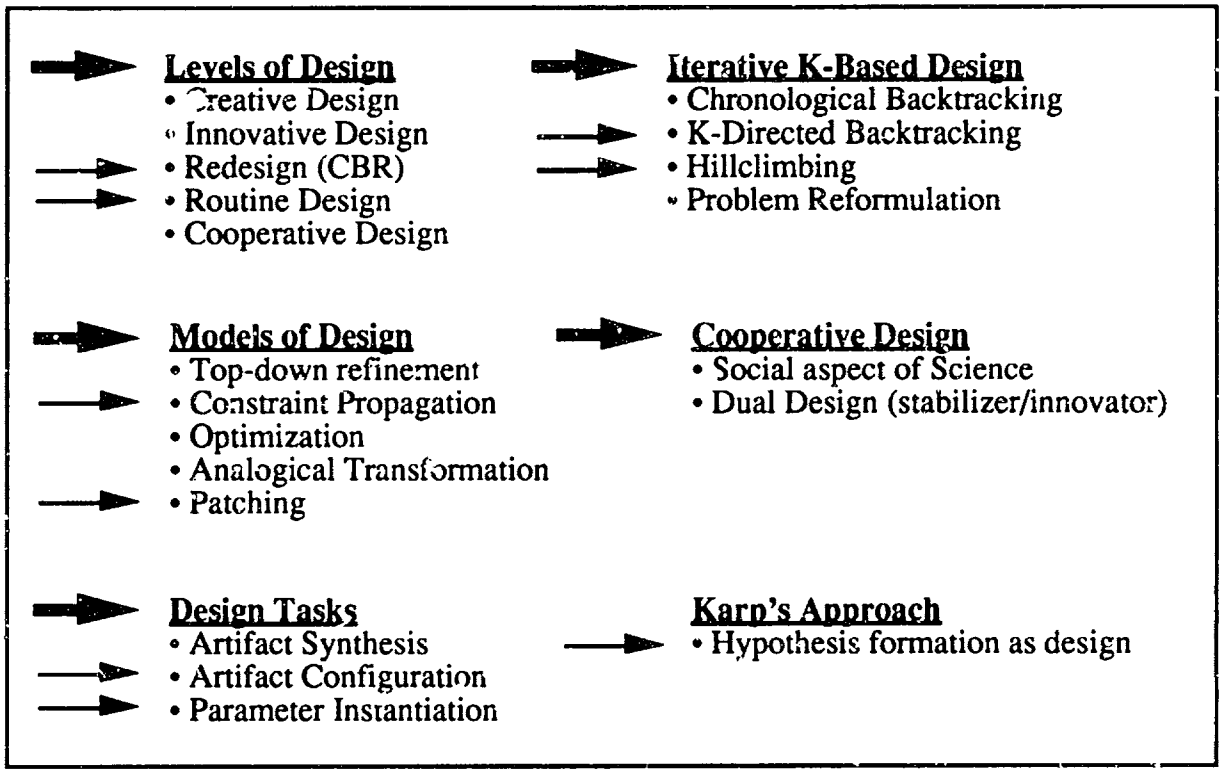


Figure 8: Applicability of design to scientific discovery

Following the classification of design tasks as described in [Groleau 1989], Table 8 compares the levels of design and scientific discovery. MARIKA performs some redesign tasks as it modifies the current scientific model structurally. It also performs routine science when modifying parameter assignments. As explained earlier, scientific discovery is an iterative process requiring large amounts of knowledge, whether the domain is new and broad or older and narrower. MARIKA performs a simple form of knowledge-directed backtracking when using table lookup to suggest a structural patch or temporarily ignoring a range constraint. My system performs hill climbing at a high level,

considering modeling bugs in succession; it also uses hill climbing when adjusting parameter boundaries.

Design Level	Problem Solving Technique	Discovery Level	Discovery System
Creative design	?	Historical discoveries	KEKADA, IDS, etc...
Innovative design	Case-based Reasoning, Analogy, Commutation, etc...	Qualitative & Quantitative laws	BACON, FAHRENHEIT, etc...
Redesign	Expert Systems, Case-based Reasoning, Parameter Optimization, etc...	Model-based scientific discovery	GORDIUS, MARIKA, etc...
Routine design	Hierarchical Refinement, Constraint Propagation, etc...	Routine science = engineering	?
Cooperative design	Blackboard, Multi-agent Architecture, etc...	Scientific community	?

Table 8: Comparison of design and scientific discovery levels.

Most models of design are applicable to scientific discovery. GORDIUS performs top-down refinement of hypotheses as it tests generic hypotheses before attempting to apply them in specific instances. MARIKA uses constraint propagation and patching techniques[16]. All design tasks are performed by scientific theory design systems. For example, GORDIUS first synthesizes a theory before debugging it, whereas MECHEM/STOICH and MARIKA configure a theory and instantiate parameters if

[16]Patching techniques are also used in GORDIUS, GENSIM, and MECHEM/STOICH.

necessary. Discovery systems do not model the cooperative aspect of science. The historical data contained in the great re-discovery systems, such as KEKADA, IDS, GENSIM, or CER, often contain data gathered by many scientists working in different labs. Comprehensive discovery systems of the future may consider distributed architectures where each level of discovery can be addressed by a specialized module. These modules could share discoveries through a blackboard-like scheduling system.

IV.3.4. Conclusion

MARIKA implements the last three steps of model-based discovery in an autonomous fashion. Data are provided by the experimenter and MARIKA has no influence on the data set it is provided with. Diagnosis techniques are used to compare the data to the model predictions and to identify the discrepancies. Fixing the discrepancies requires simple design techniques akin to redesign (*a priori* countable structural modifications) or routine design (parameter instantiation).

CHAPTER 5

Working Examples

We expect them [theories] to improve as did the maps of the world; we hope for exciting new discoveries from time to time, and we can count upon steady cleansing and incremental improvement in detail. But one thing we ought not to expect: with our theories, our maps of the processes of the universe, we should not expect to make simple pictures of what was never visual in the first place.
But we will always try. [Morrison and Morrison 1987]

V. Demonstration of the System

The purpose of this section is to demonstrate the proposed system using simple examples. I used a rotating chair stimulus and a "barbecue spit" stimulus, as explained below. The rotating chair tests the Vestibulo-Ocular Reflex (VOR). Subjects are seated erect in a chair with eyes open in the dark. Their limbs and head are strapped to the chair. They wear long sleeves and pants thereby minimizing tactile cues. Because they are spun around a vertical axis, the horizontal semicircular canals are the only vestibular organs stimulated[17]. Horizontal Eye movements in the dark are recorded in order to measure the compensatory response to the velocity estimate of the CNS. Conditions are similar for the barbecue spit except that subjects are rotated around an earth horizontal axis through their long body axis. In this case, the otoliths are stimulated, by the varying gravity component, as well as the horizontal semicircular canals.

My system is able to tune the model for normal subjects as well as discover pathological cases with unilateral semicircular canal deficiency, central nervous lesions affecting the VOR time constant, bilateral reduced otolith response, and hyperactive semicircular canal-otolith interaction.

Such examples are simple enough for someone with a rudimentary understanding of the vestibular system to discover the pathological cases. They demonstrate the behavior of the system as well as interesting concepts. For example, in the case of unilateral deficiency, the system should hypothesize that either the left canal is inoperative and the

[17]The horizontal canals represent a simplification from reality where the horizontal canals are pitched up significantly. Another argument could be that the otoliths placed on each side of the head are stimulated by the centrifugal force. Assuming the distance from them to the rotational axis is 3 cm (a little over an inch), such force would remain below otolith detection threshold at speeds up to 130 degrees/s. Moreover, these stimuli would indicate accelerations in opposite directions and the CNS would probably cancel them out.

right one functions correctly or the right canal is inoperative and the left canal behaves like a right canal. The first solution is clearly chosen for its parsimony.

V.1. Data Sources

The data necessary for the above demonstration were obtained from the studies performed in [Furman 1989, Furman et al. 1990, Wall et al. 1984, Wall et al. 1989, Wall et al. 1984, Wall et al. 1987, Wall 1990] and from the Vestibular Laboratory at the Massachusetts Eye and Ear Infirmary (MEEI).

V.2. Rotating Chair Examples

I will first use the rotating chair stimulus just described. The stimulus occurring entirely in the earth horizontal plane, I restrict my calculations to the direction perpendicular to that plane in which all signals involved can be expressed. I will first show that the system is able to tune the model for normal subjects as well as to discover unilateral semicircular canal deficiency. I will also demonstrate tuning of the system for abnormally low semicircular canal sensitivity.

As described in Chapter 4, the system can be summarized as executing the following steps:

1) Introduction of the model parameters range constraints.
2) Simulation of the hypothesized model.

3) Fit of all signals present in the model, including inputs and outputs.
4) Generalization of the model by parameterizing the curve fits.
5) Introduction of the constraints derived from initial experimental conditions.
6) Introduction of the constraints derived from differentiation relations between the signals.
7) Introduction of the constraints derived from the model equations.
8) Estimation of the curve fit parameter of the experiment data.
9) Propagation of all constraints.

Where step 9) has three possible outcomes:

a) No violation: the system has produced a set of constraints that prove that the individual's vestibular behavior is adequately represented by the current model.
b) Structural constraint violation: the system must look up corresponding structural patches, build the modified models and start the process over.
c) Variable range constraint violation: the system must relax model parameters constraints until all variable range constraints are satisfied.

Figure 9 shows the model as represented in Extend. Notice the existence of two semicircular canals. Their outputs are merged using weights that depend on the direction of rotation. The CNS has a single representation of the semicircular canal sensors.

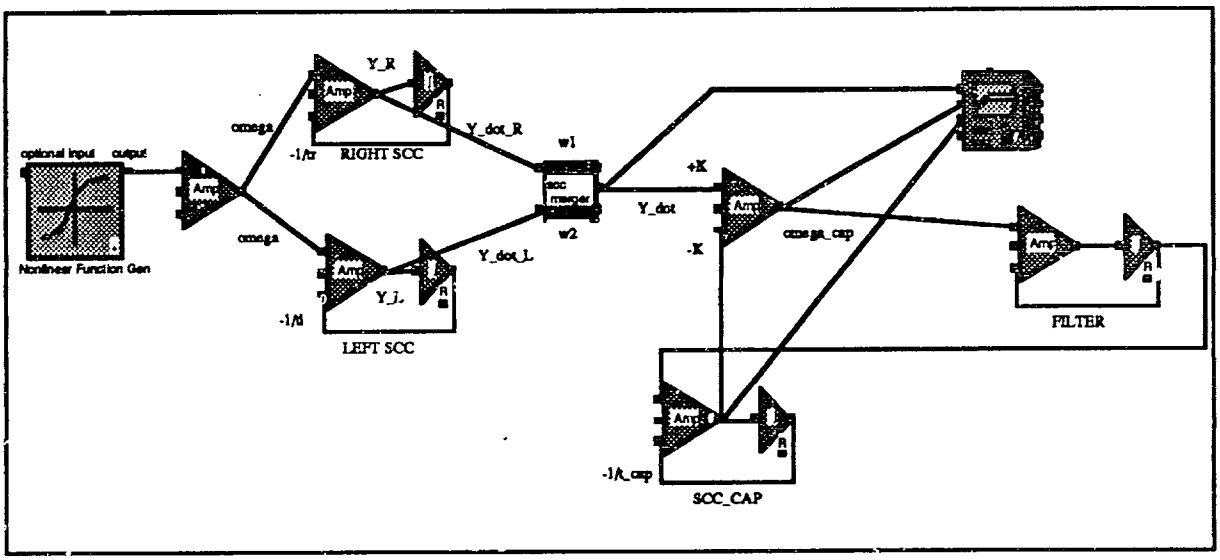


Figure 9: The Extend model for the rotating chair

The corresponding model equations are:

$$\text{eq.1: } \dot{Y}_R = \omega - \frac{Y_R}{\tau_R}$$

$$\text{eq.2: } \dot{Y}_L = \omega - \frac{Y_L}{\tau_L}$$

$$\text{eq.3: } \dot{Y} = w_1 * \dot{Y}_R + w_2 * \dot{Y}_L \text{ OR } \dot{Y} = w_2 * \dot{Y}_R + w_1 * \dot{Y}_L$$

$$\text{eq.4: } \hat{\dot{Y}} = \hat{\omega} - \frac{\hat{Y}}{\hat{\tau}}$$

$$\text{eq.5: } \hat{\omega} = k\left(\dot{Y} - \hat{\dot{Y}}\right)$$

Notice that eq.3 exists in two versions, one for clockwise rotation and one for counterclockwise rotation.

I simulated this model using the input pictured in Figure 10-A (ω) and obtained the output pictured in Figure 10-B ($\hat{\omega}$).

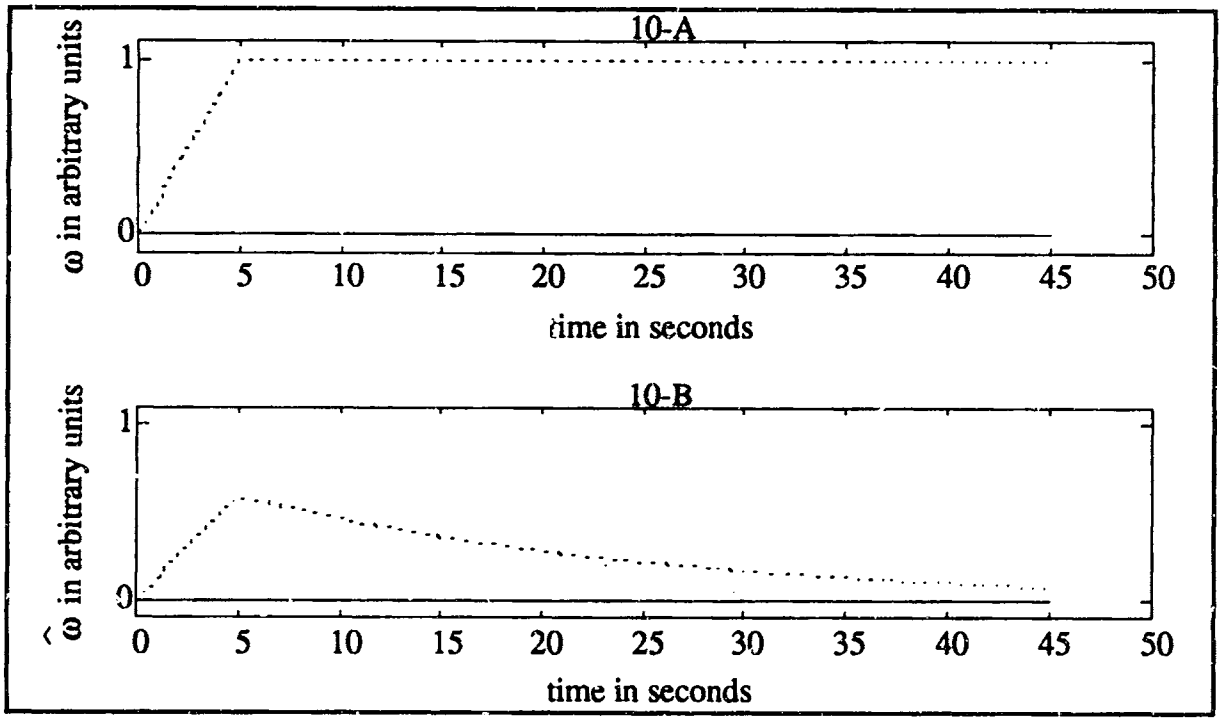


Figure 10: Velocity ramp input and estimated output velocity

The model produces the internal signals pictured in Figure 11. All velocities are in arbitrary units. Figure 11-A shows Y_R (or Y_L), Figure 11-B shows $\dot{Y}$ (or $\dot{Y}_R$, or $\dot{Y}_L$), Figure 11-C shows $\hat{Y}$ and Figure 11-D shows $\hat{\dot{Y}}$.

During the first segment, the seven signals can be represented by the following shapes:

$$\omega = 0.2 * t$$

$$Y_R = -9.8 * \left(1 - e^{-t/7.0}\right) + 1.4 * t$$

$$\dot{Y}_R = 1.4 - 1.4 * e^{-t/7.0}$$

$$Y_L = -9.8 * \left(1 - e^{-t/7.0}\right) + 1.4 * t$$

$$\dot{Y}_L = 1.4 - 1.4 * e^{-t/7.0}$$

$$\dot{Y} = 1.4 - 1.4 * e^{-t/7.0}$$

$$\hat{\omega} = 2.8 * \left(1 - e^{-t/21.0}\right)$$

$$\hat{Y} = -9.8 * \left(1 - e^{-t/7.0}\right) + 29.4 * \left(1 - e^{-t/21.0}\right)$$

$$\hat{\dot{Y}} = -1.4 * e^{-t/7.0} + 1.4 * e^{-t/21.0}$$

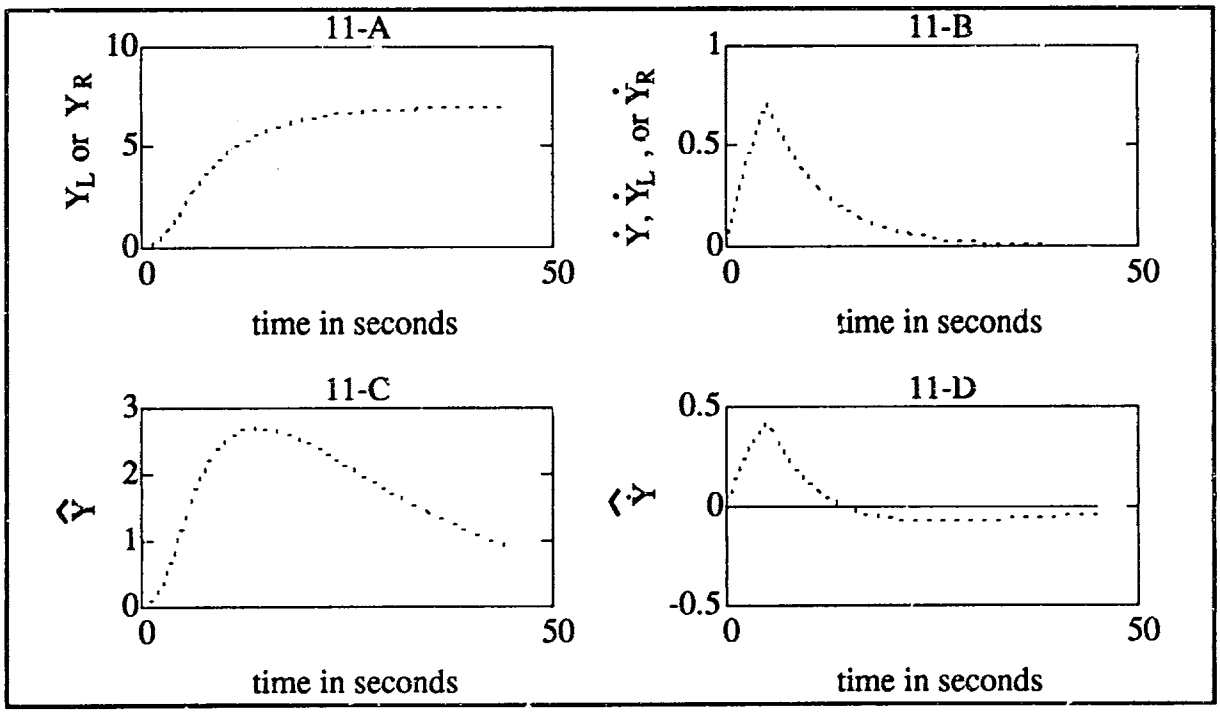


Figure 11: Simulated internal signals

Let us focus on the first signals, ω, Y_R, and $\dot{Y}_R$. The generalizations of the signals are:

$$\omega = A * t$$

$$Y_R = B + C * t + D * e^{-t/\tau_1}$$

$$\dot{Y}_R = E + F * e^{-t/\tau_2}$$

Differentiation constraints dictate that:

$$\frac{dY_R}{dt} = \dot{Y}_R$$

OR

$$C - \frac{D}{\tau_1} * e^{-t/\tau_1} = E + F * e^{-t/\tau_2}$$

OR

$$C = E$$
$$t_1 = t_2$$
$$-\frac{D}{\tau_1} = F$$

Initial conditions constraints dictate that:

$$@\ t = 0 : Y_R = 0,\ \dot{Y}_R = 0$$

OR

$$B + D = 0$$
$$E + F = 0$$

After constraint propagation, the two generalized signals are correctly represented as:

$$Y_R = B_R * \left(1 - e^{-t/\tau_R}\right) - \frac{B_R}{\tau_R} * t$$

$$\dot{Y}_R = -\frac{B_R}{\tau_R} * \left(1 - e^{-t/\tau_R}\right)$$

Using the same techniques, the generalizations of the other signals are represented as:

$$\omega = A * t$$

$$Y_L = G + H * t + I * e^{-t/\tau_3}$$

$$\dot{Y}_L = J + KK * e^{-t/\tau_4}$$

$$\dot{Y} = L + M * e^{-t/\tau_5}$$

$$\hat{\omega} = N + O * e^{-t/\tau_6}$$

$$\hat{Y} = P + Q * e^{-t/\tau_7} + R * e^{-t/\tau_8}$$

$$\hat{\dot{Y}} = S * e^{-t/\tau_9} + T * e^{-t/\tau_{10}}$$

The model equations give[18]:

$$\text{eq.1: } A - \frac{C}{\tau_R}, E = - \frac{B}{\tau_R}, F = - \frac{D}{\tau_R}, \tau_1 = \tau_2$$

$$\text{eq.2: } A - \frac{H}{\tau_L}, J = - \frac{G}{\tau_L}, KK = - \frac{I}{\tau_L}, \tau_3 = \tau_4$$

$$\text{eq.3: } L = w_1 * E + w_2 * J, \; M = w_1 * F + w_2 * KK, \; \tau_2 = \tau_4 = \tau_5$$

$$\text{eq.4: } N - \frac{P}{\hat{\tau}} = 0, S - \frac{Q}{\hat{\tau}} = 0, T = N - \frac{R}{\hat{\tau}} + O, \tau_7 = \tau_9, \tau_6 = \tau_8 = \tau_{10}$$

$$\text{eq.5: } N = k * L, O = - k * T, k * M - k * S = 0, \; \tau_5 = \tau_9, \tau_6 = \tau_{10}$$

The differentiation equations give:

$$C = E \cdot - \frac{D}{\tau_1} = F, \; \tau_1 = \tau_2$$

$$H = J, - \frac{I}{\tau_3} = KK, \; \tau_3 = \tau_4$$

$$- \frac{Q}{\tau_7} = S, - \frac{R}{\tau_8} = T, \; \tau_7 = \tau_9, \; \tau_8 = \tau_{10}$$

I will demonstrate the system using this model under three different sets of circumstances. The constraint network described above will remain identical for all three.

V.2.1. Normal Continuous Rotating Chair

First I assume a normal experiment. Let's assume the curve fit variables to be:

$$A = 0.2$$
$$\tau_6 = 21.0$$
$$N = 2.8$$
$$O = - 2.8$$

[18]The first version of eq.3 is used in the example.

Assuming the following model parameter constraints,

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 1.00]$$
$$w_2 = [0.17 ; 1.00]$$
$$k = [1.0 ; 3.0]$$

the system provides the following constraint set:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 0.79]$$
$$w_2 = [0.17 ; 0.79]$$
$$k = [1.22 ; 2.50]$$

in which w1, w2 and k are further constrained.

V.2.2. Single Canal

Next, I will assume a deficient right semicircular canal. In this setting, the curve fit variables are assumed to be:

Clockwise rotation

$$A_1 = 0.2$$

$$\tau_{61} = 21.0$$

$$N_{61} = - O_{61} = 1.87 \left(\frac{2}{3} \text{ of nominal value} \right)$$

Counterclockwise rotation

$$A_2 = 0.2$$

$$\tau_{62} = 21.0$$

$$N_{62} = - O_{62} = 0.93 \left(\frac{1}{3} \text{ of nominal value} \right)$$

If simulated independently, those two experiments provide conflicting but internally consistent constraint sets:

Clockwise rotation

$$\tau_R = [6.0 ; 8.0]$$

$$\tau_L = [6.0 ; 8.0]$$

$$\hat{\tau} = [6.0 ; 8.0]$$

$$w_1 = [0.17 ; 0.79]$$

$$w_2 = [0.17 ; 0.79]$$

$$k = [1.62 ; 2.5]$$

Counterclockwise rotation

$\tau_R = [6.0 ; 8.0]$

$\tau_L = [6.0 ; 8.0]$

$\hat{\tau} = [6.0 ; 8.0]$

$w_1 = [0.17 ; 0.31]$

$w_2 = [0.17 ; 0.31]$

$k = [1.62 ; 2.33]$

The conflict does not appear among the model parameters. However four curve fit parameters exhibit incompatibilities:

Clockwise rotation

$L_1 = [0.75 ; 1.15]$

$P_1 = [11.20 ; 14.93]$

$R_1 = [-24.12 ; -15.68]$

$T_1 = [0.75 ; 1.15]$

Counterclockwise rotation

$L_2 = [0.40 ; 0.57]$

$P_2 = [5.60 ; 7.47]$

$R_2 = [-12.06 ; -8.40]$

$T_2 = [0.40 ; 0.57]$

When simulated consecutively within the same network that comprises two sets of curve fit variables and a single set of model parameters, the system fails during propagation indicating:

```
Failed when attempting to set max of
  (+ (* W2 E2) (* W1 J2)) to 0.61
```

The constraint violated is clearly the second version of the one derived from eq.3 that was formulated as:

$$L = w_2 * E + w_1 * J$$

This constraint was posted as a result of the second experiment, as the curve fit variables indices are set to 2 (E2 and J2) and the equation reflects a counterclockwise rotation. At this point, the model parameters set is:

$$\tau_R = [6.00 ; 8.00]$$
$$\tau_L = [6.00 ; 8.00]$$
$$\hat{\tau} = [7.27 ; 8.00]$$
$$w_1 = [0.34 ; 1.00]$$
$$w_2 = [0.17 ; 0.31]$$
$$k = [1.62 ; 1.79]$$

The patch table lookup provides two possible model patches. The first one corresponds to a deficient right canal, the second one corresponds to a deficient left canal.
The first patch requires the model equations to be modified as follows:

$$\text{eq.2: } \dot{Y}_L = \omega - \frac{Y_L}{\tau_L}$$

$$\text{eq.3: } \dot{Y} = w_2 * \dot{Y}_L \text{ OR } \dot{Y} = w_1 * \dot{Y}_L$$

$$\text{eq.4: } \hat{\dot{Y}} = \hat{\omega} - \frac{\hat{Y}}{\hat{\tau}}$$

$$\text{eq.5: } \hat{\omega} = K\left(\dot{Y} - \hat{\dot{Y}}\right)$$

This patch is successful of course and results in the following set of model parameters:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.23 ; 0.48]$$
$$w_2 = [0.47 ; 0.96]$$
$$k = [1.62 ; 2.5]$$

Interestingly, the second patch is also successful and results in the following set of model parameters:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.47 ; 0.96]$$
$$w_2 = [0.23 ; 0.48]$$
$$k = [1.62 ; 2.5]$$

This model assumes the remaining right canal behaves as a left canal and has a stronger sensitivity in the clockwise direction. The first patch is chosen by the system because it involves a single change, while the second patch involves two changes.

V.2.3. Abnormally Low Canal Sensitivity

Next, I will show an example where the model is structurally correct but some of the limits of the model parameters have to be extended beyond their normal range.

Let's assume the curve fit variables to be:

$$A = 0.2$$
$$\tau_6 = 21.0$$
$$N = 0.93$$
$$O = -0.93$$

and the model parameter constraints to be:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 0.67]$$
$$w_2 = [0.33 ; 1.33]$$
$$k = [1.0 ; 3.0]$$

The system fails during propagation indicating:

```
Failed when attempting to set the min of W2 to 0.33
```

This constraint isn't derived from an equation. Moreover, the problem occurred while implementing the upper bound of the w2 parameter. The system is therefore run ignoring the upper bound on w2. All troublesome model parameter constraints are removed until the remaining system can be satisfied. In this case no other limit has to be suspended.

The resulting constraint set without a lower limit on w2 is:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 0.67]$$
$$w_2 = [-0.58 ; 0.31]$$
$$k = [1.62 ; 2.50]$$

The first limit to relax is the 0.33 lower bound on w2. This produces a consistent system resulting in the following parameter set:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 0.48]$$
$$w_2 = [0.00 ; 0.31]$$
$$k = [1.62 ; 2.50]$$

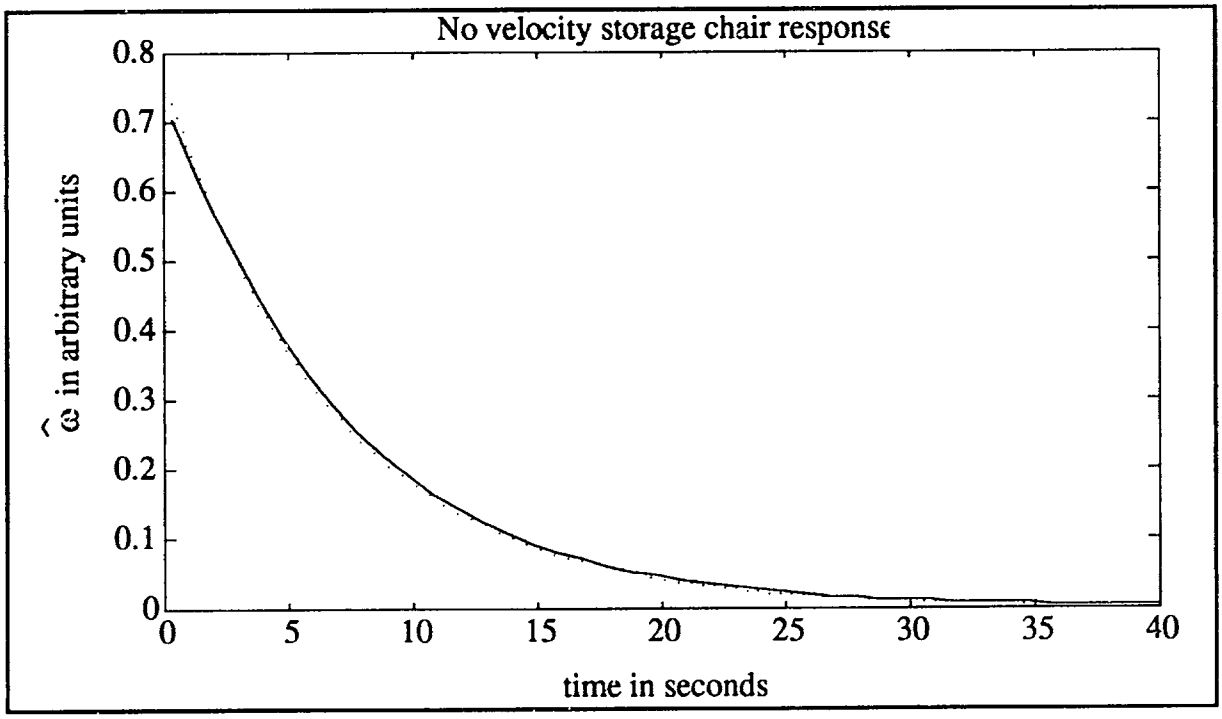
If I choose to modify the lower bound on w1 instead, the system also reaches equilibrium, although a different one:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.00 ; 0.15]$$
$$w_2 = [0.33 ; 0.48]$$
$$k = [1.62 ; 2.33]$$

V.2.4. Absence of Velocity Storage

The next example involves simulated chair data for a patient with no velocity storage. Velocity storage is the property of the vestibular system to maintain a sensation of rotation long after the canal afferent signal has decayed. In my model, velocity storage is represented by the existence of the CNS internal model and the feedback gain. When velocity storage is absent, the time constant of the estimated velocity is the same as the afferent time constant. The feedback loop of the observer model is cut to predict data correctly.

The second segment of the output and the associated curve fit are:



with an exponential decay fit of

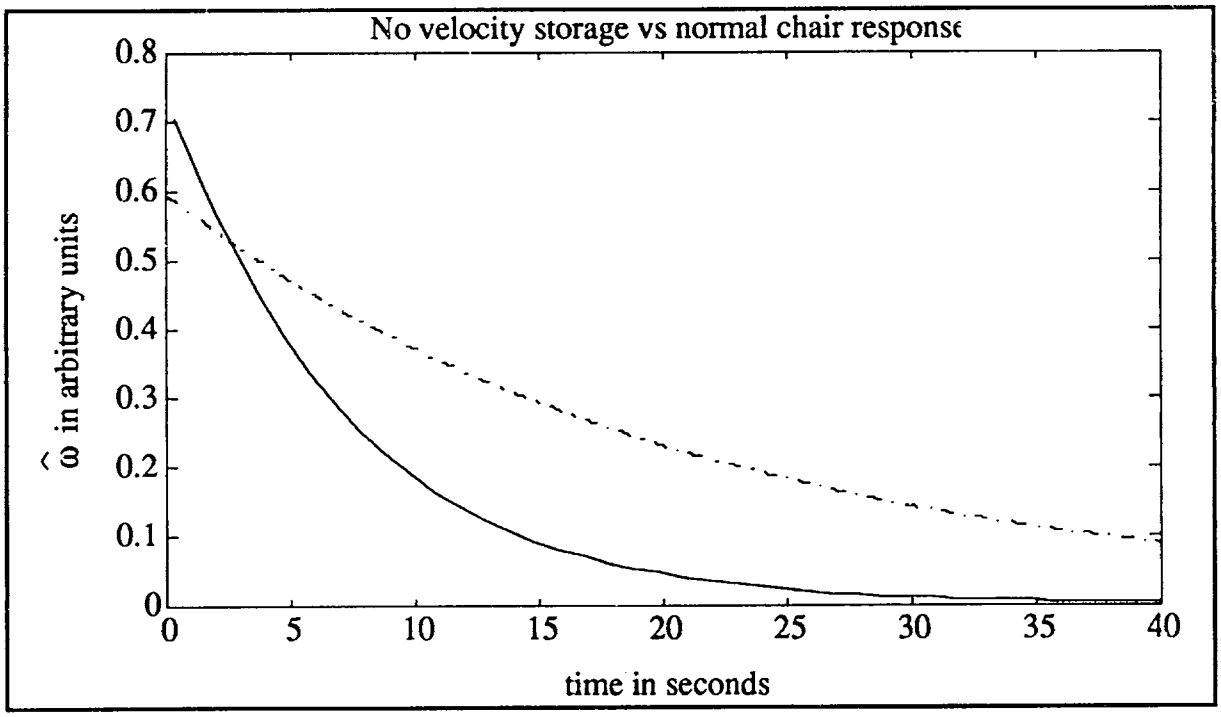
$$f(t) = A * e^{(-t/T)}$$

where

$T = 7.22$ (samples)

$A = 0.74$ (arbitrary units)

By comparison, the normal response has a lower maximum but a longer time constant:



When presented with these data, MARIKA first unsuccessfully tries to relax the lower bound of k. However, the value would have to be lowered to 0.00 to obtain a consistent constraint network. Because a value of 0.00 for the feedback gain parameter indicates no feedback, the parameter k cannot be lowered and the patch table[19] proposes to restructure the model without the feedback loop, which is equivalent to destroying the velocity storage mechanism.

[19]The patch table is pre-compiled from clinical knowledge.

Once the model has been modified in this way, the system easily satisfies the constraints and returns an unmodified set of model parameters:

$$\tau_R = [6.0 ; 8.0]$$
$$\tau_L = [6.0 ; 8.0]$$
$$\hat{\tau} = [6.0 ; 8.0]$$
$$w_1 = [0.17 ; 0.67]$$
$$w_2 = [0.33 ; 1.33]$$
$$k = [1.0 ; 3.0]$$

V.2.5. Normal Sinusoidal Rotating Chair

The next example shows steady state sinusoidal chair data for a normal human. This data set was obtained from MEEI. The chair velocity is sampled at 10 Hz with a sinusoidal fit of

$$f(t) = A * \sin(t/T + P) + C$$

where

T = 3.18 (samples/radian)

A = -817 (arbitrary units)

P = -0.36 (radians)

C = 68.02[20] (arbitrary units)

The frequency[21] of the chair velocity is given by:

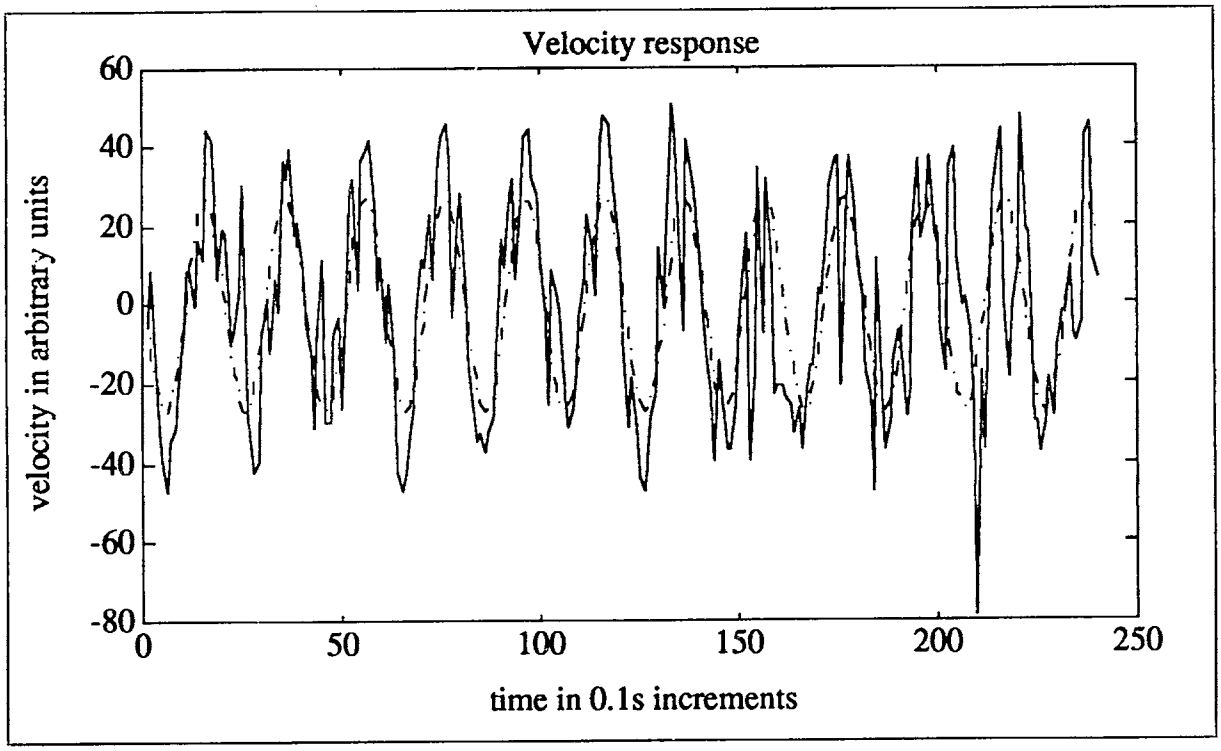
$$\omega = \frac{10}{T * 2 * \pi} = 0.500 \text{ Hz}$$

[20]This offset is an artifact due to imperfect calibration of the null position.

[21]The factor 10 in the numerator reflects the fact that there are 10 data points per second.

The chair velocity is not calibrated; however, the maximum chair velocity is known to be 50 degrees per second.

The velocity estimate measured from desaccaded slow phase velocity of eye movements was sampled at 10 Hz and is given in solid line by:



with a dotted sinusoidal fit of

$$f(t) = A * \sin(t/T + P) + C$$

where

T = 3.20 (samples/radian)

A = 26.86 (arbitrary units)

P = 2.93 (radians)

C = 0.00 (arbitrary units)

The frequency[22] of the response is given by:

[22]The factor 10 in the numerator reflects the fact that there are 10 data points per second.

$$\omega = \frac{10}{T * 2 * \pi} = 0.497 \text{ Hz}$$

The signals can be represented as:

$$\omega = A * \sin(t/T + B)$$

$$Y = C * \sin(t/T + C)$$

$$\dot{Y} = E * \sin(t/T + F)$$

$$\hat{\omega} = G * \sin(t/T + H)$$

$$\hat{Y} = I * \sin(t/T + J)$$

$$\hat{\dot{Y}} = L * \sin(t/T + M)$$

Assuming a single semicircular canal to simplify the model, the model equations are:

$$\text{eq.1: } \dot{Y} = \omega - \frac{Y}{\tau}$$

$$\text{eq.2: } \hat{\dot{Y}} = \hat{\omega} - \frac{\hat{Y}}{\hat{\tau}}$$

$$\text{eq.3: } \hat{\omega} = k\left(\dot{Y} - \hat{\dot{Y}}\right)$$

These equations are translated into the following constraints:

$$E = \sqrt{\left(A * \cos B - \frac{C}{\tau} * \cos D\right)^2 + \left(A * \sin B - \frac{C}{\tau} * \sin D\right)^2}$$

$$F = \arctan\left(\frac{A * \sin B - \frac{C}{\tau} * \sin D}{A * \cos B - \frac{C}{\tau} * \cos D}\right)$$

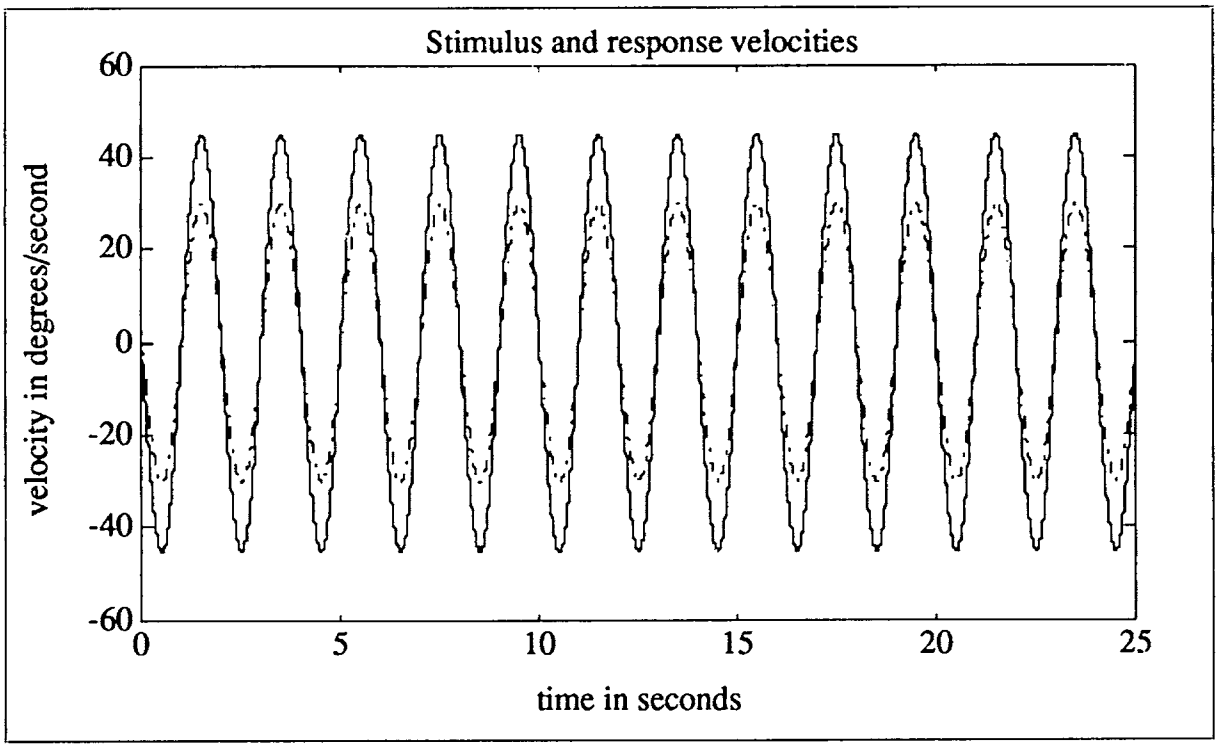
$$L = \sqrt{\left(G * \cos H - \frac{I}{\hat{\tau}} * \cos J\right)^2 + \left(G * \sin H - \frac{I}{\hat{\tau}} * \sin J\right)^2}$$

$$F = \arctan\left(\frac{G * \sin H - \frac{I}{\hat{\tau}} * \sin J}{G * \cos H - \frac{I}{\hat{\tau}} * \cos J}\right)$$

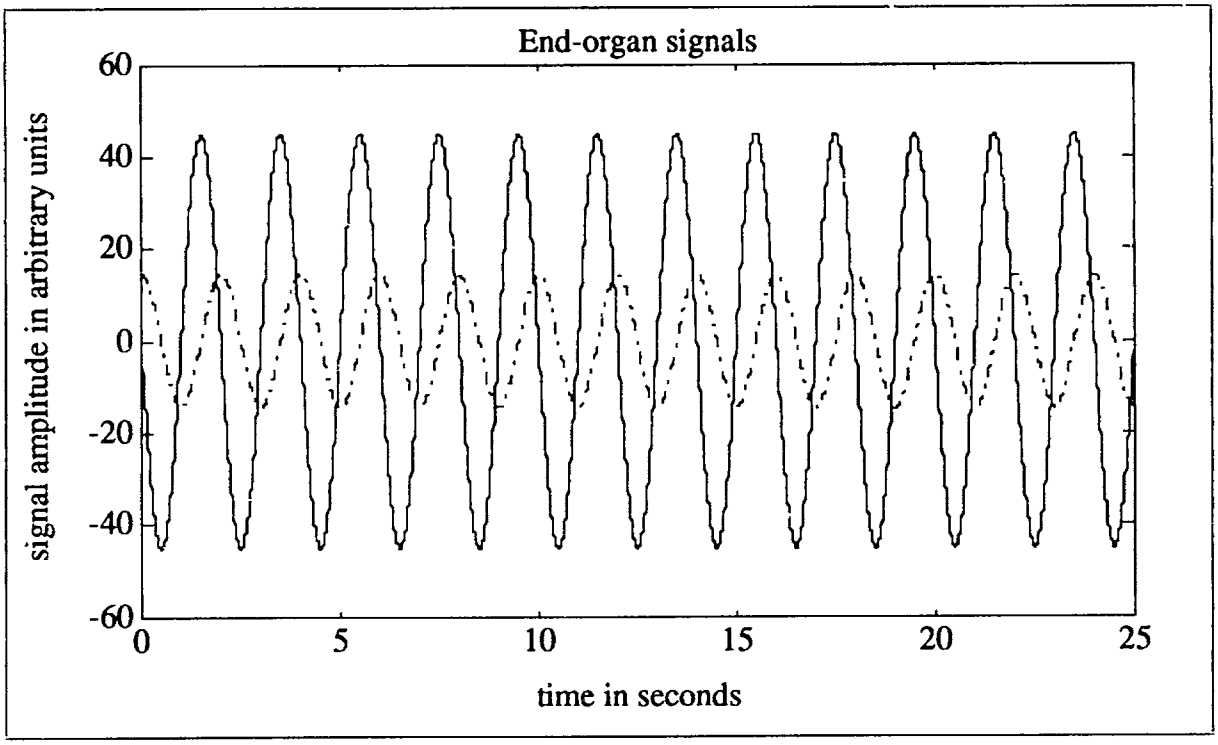
$$A = K * \sqrt{(E * \cos F - I * \cos J)^2 + (E * \sin F - I * \sin J)^2}$$

$$B = \arctan\left(\frac{E * \sin F - I * \sin J}{E * \cos F - I * \cos J}\right)$$

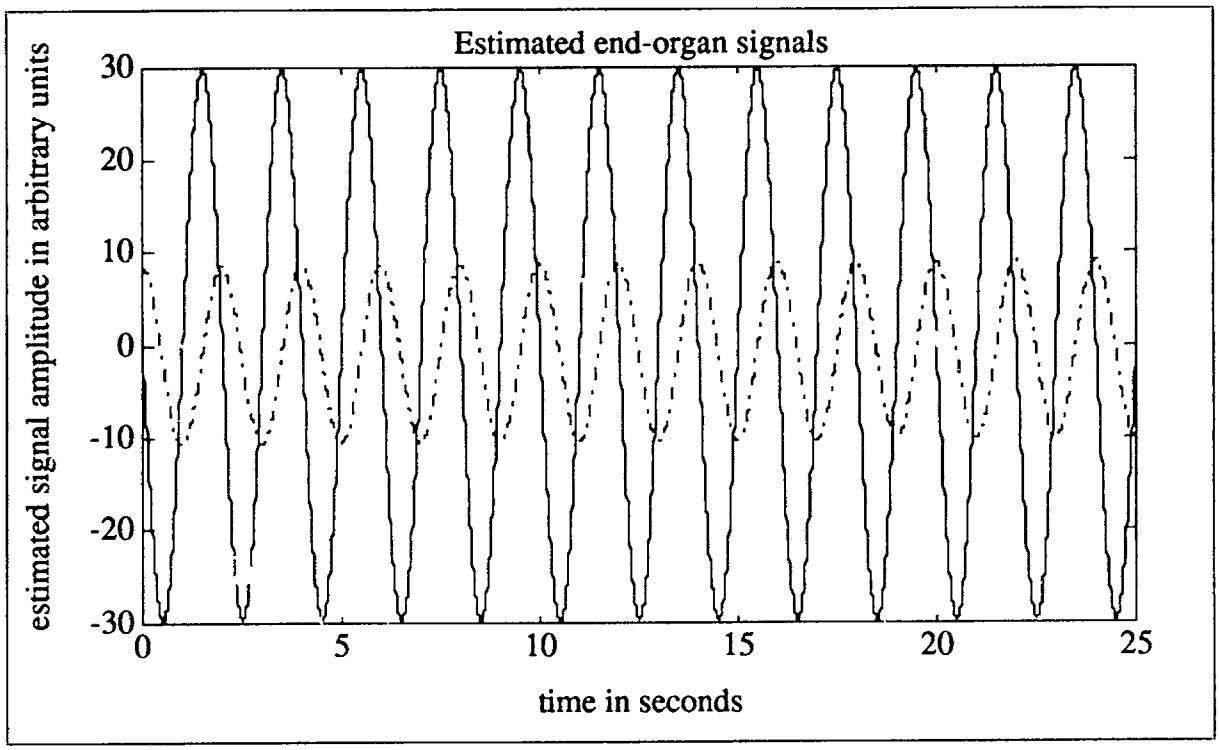
The input velocity ω (solid line) and the predicted output $\hat{\omega}$ (dotted line) are:



The intermediary signals Y (solid line) and $\dot{Y}$ (dotted line) are predicted as:



The intermediary signals $\hat{Y}$ (solid line) and $\hat{\dot{Y}}$ (dotted line) are predicted as:



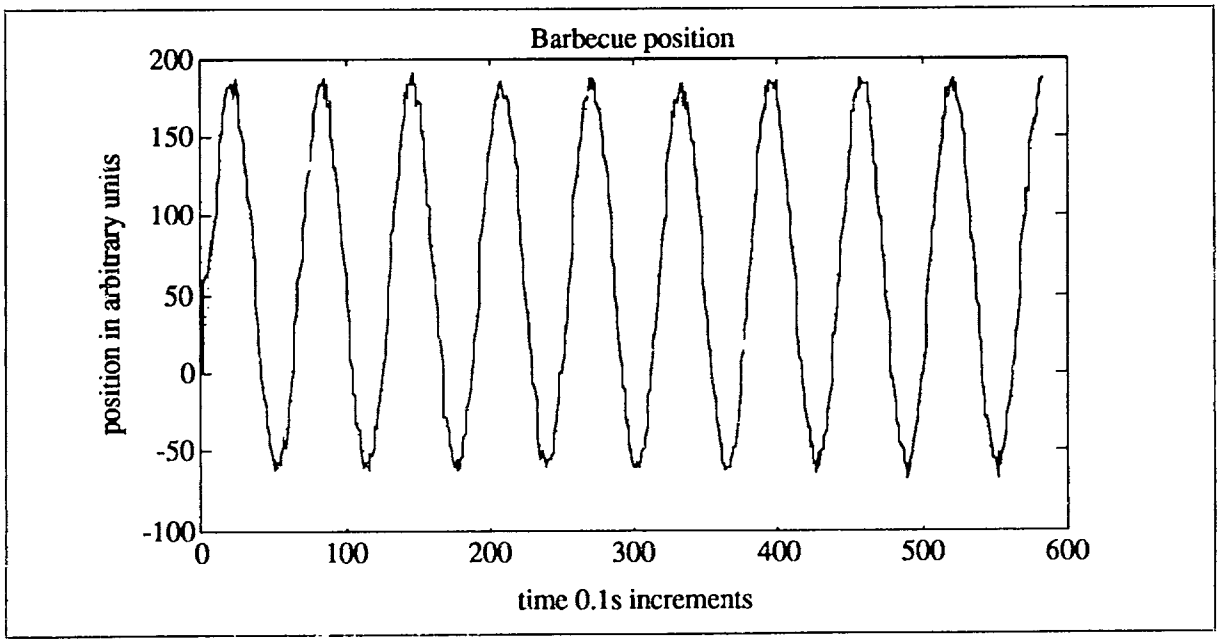
The model prediction and the data are close enough that constraint propagation does not constrain the model parameters any further. This only proves that MARIKA couldn't help refine the problem any further. To ensure that the data modeling is correct, the cross-correlation coefficient between the fitted experimental data and the model prediction can be calculated. For this data set it has a value of 0.961

V.3. Barbecue Spit Examples

I will first demonstrate the system using normal barbecue spit data from MEEI. Two simulated pathological cases follow, the first one involves canal-otolith hyper-sensitivity, while the second one demonstrates bilateral low otolith sensitivity.

V.3.1. Normal Barbecue Spit

This example shows barbecue spit data for a normal human from MEEI. The barbecue position (solid line) is sampled at 10 Hz as follows:



It is measured as the angle between a nose-down reference and the position of the subject. It has a sinusoidal fit (nearly indistinguishable dotted line) of:

$$f(t) = A * \sin(t/T + P) + C$$

where

$T = 9.94$ (samples/radian)

$A = 120.62$ (arbitrary units)

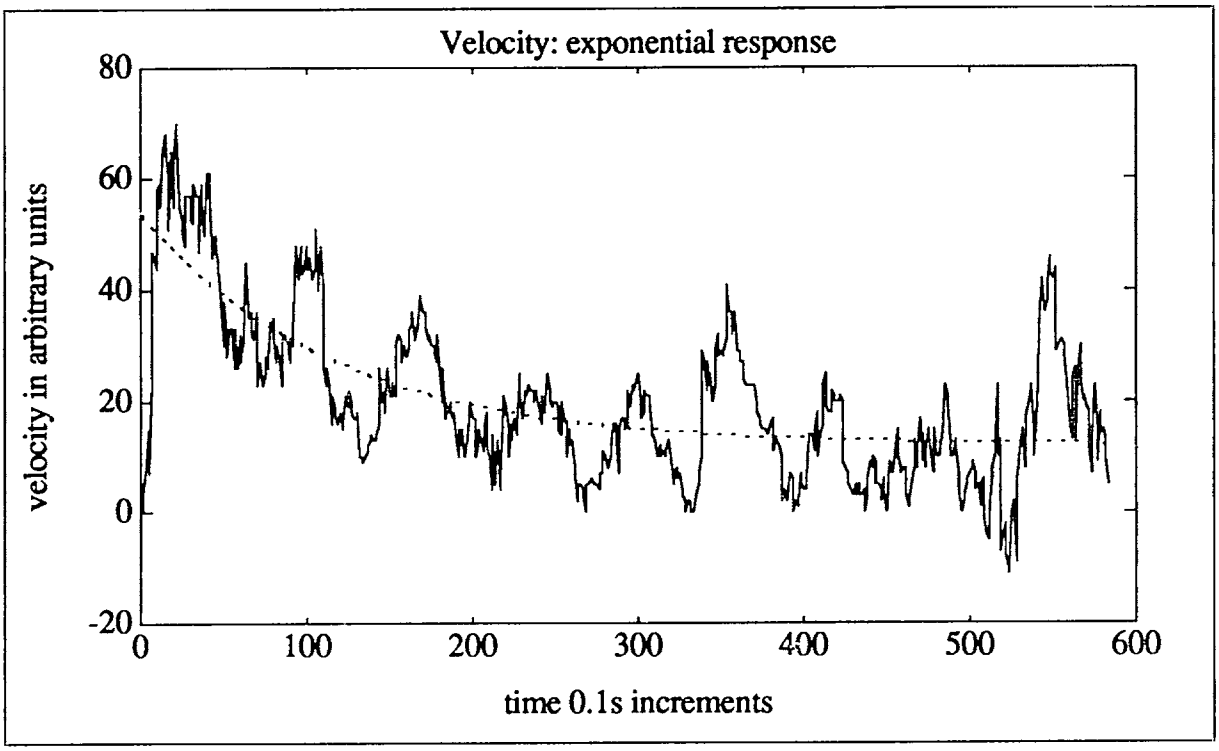
$P = -0.51$ (radians)

$C = 62.10^{23}$ (arbitrary units)

The frequency of the barbecue position and its velocity are given by:

$$\omega = \frac{10}{T * 2 * \pi} = 0.160 \text{ Hz} = 0.160 * 2 * \pi = 1.006 \text{ rad/s}$$

The velocity estimate (solid line) measured from desaccaded slow phase velocity of eye movements is:



[23]This offset is an artifact due to imperfect calibration of the null position.

with an exponential decay fit (dotted line) of

$$f(t) = A * e^{(-t/T)} + C$$

where

T = 116 (samples)

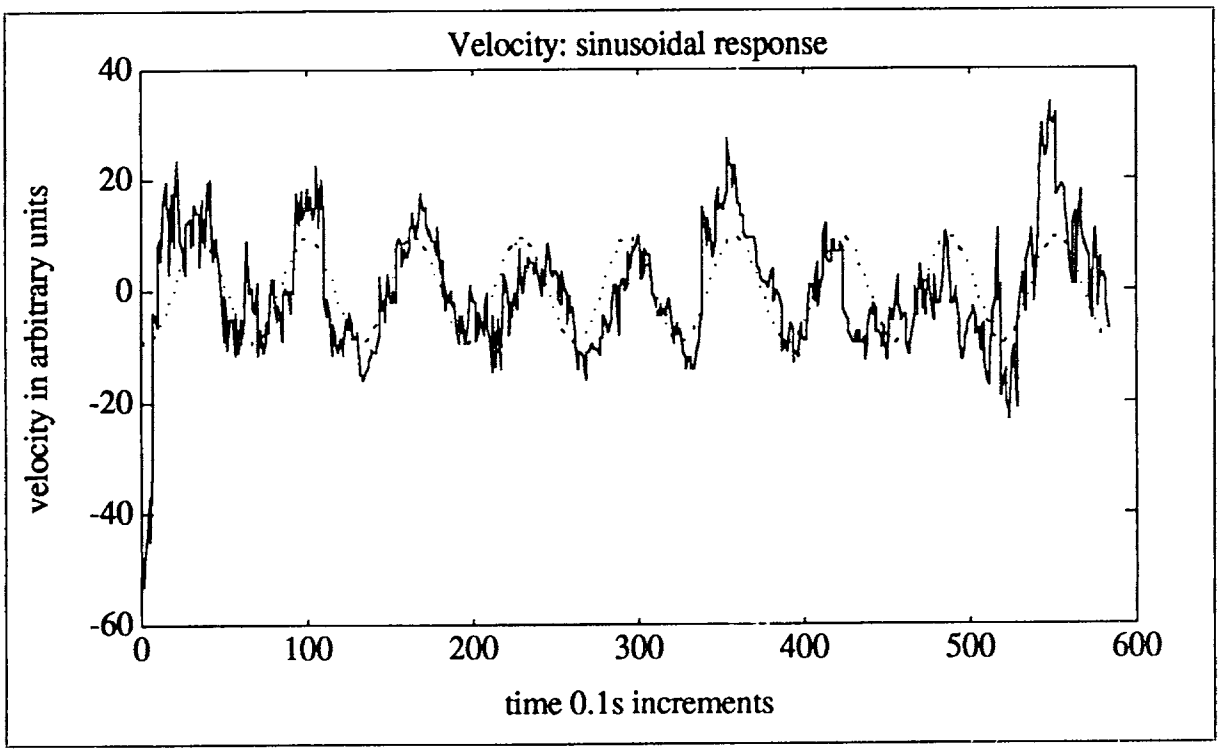
A = 41.8 (arbitrary unist)

C = 11.8 (arbitrary unist)

The time constant of the exponentially decaying portion of eye velocity is given by:

$$\tau = \frac{T}{10} = 11.60 \text{ s}$$

The peak velocity is therefore v = 41.8 + 11.8 = 53.6 °/s. Once the exponential decay fit is subtracted from the data, the remaining signal (solid line) is:



with a sinusoidal fit (dotted line) of

$$f(t) = A * \sin(t/T + P) + C$$

where

$T = 10.28$ (samples/radian)

$A = 9.45$ (arbitrary unist)

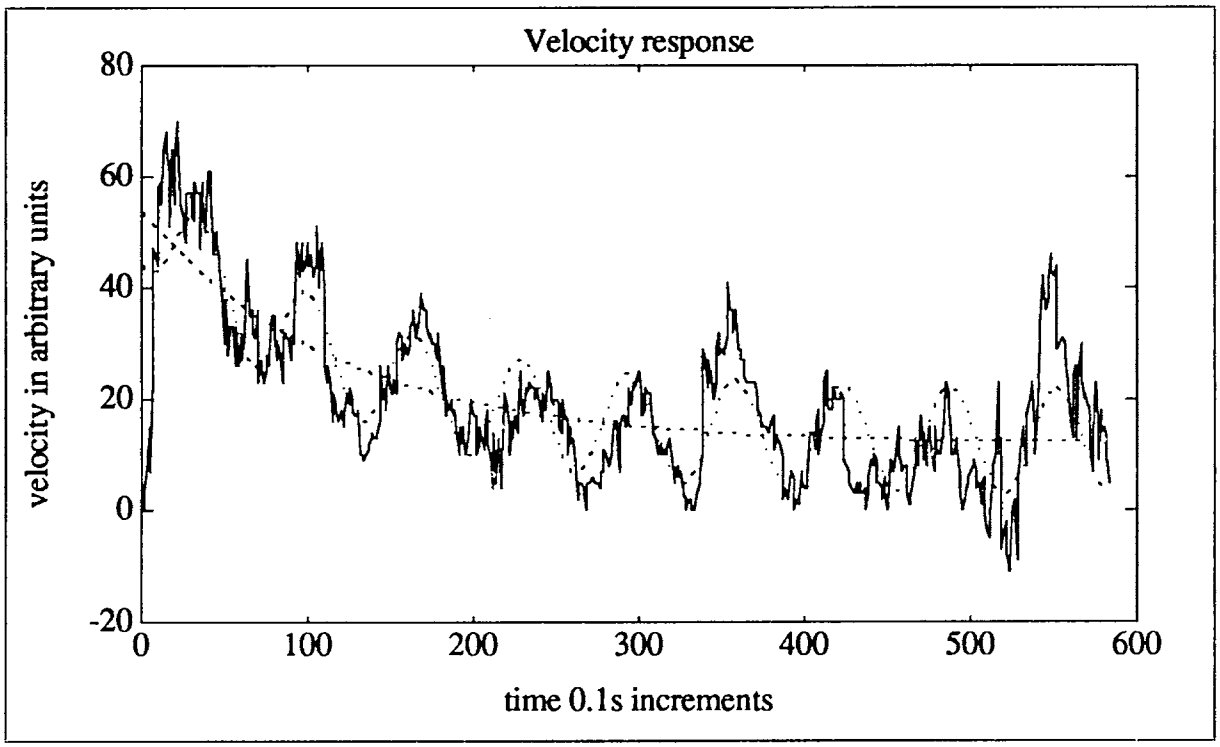
$P = -1.82$ (radians)

$C = 0.00$ (arbitrary unist)

The frequency of the sinusoidal portion of eye velocity is given by:

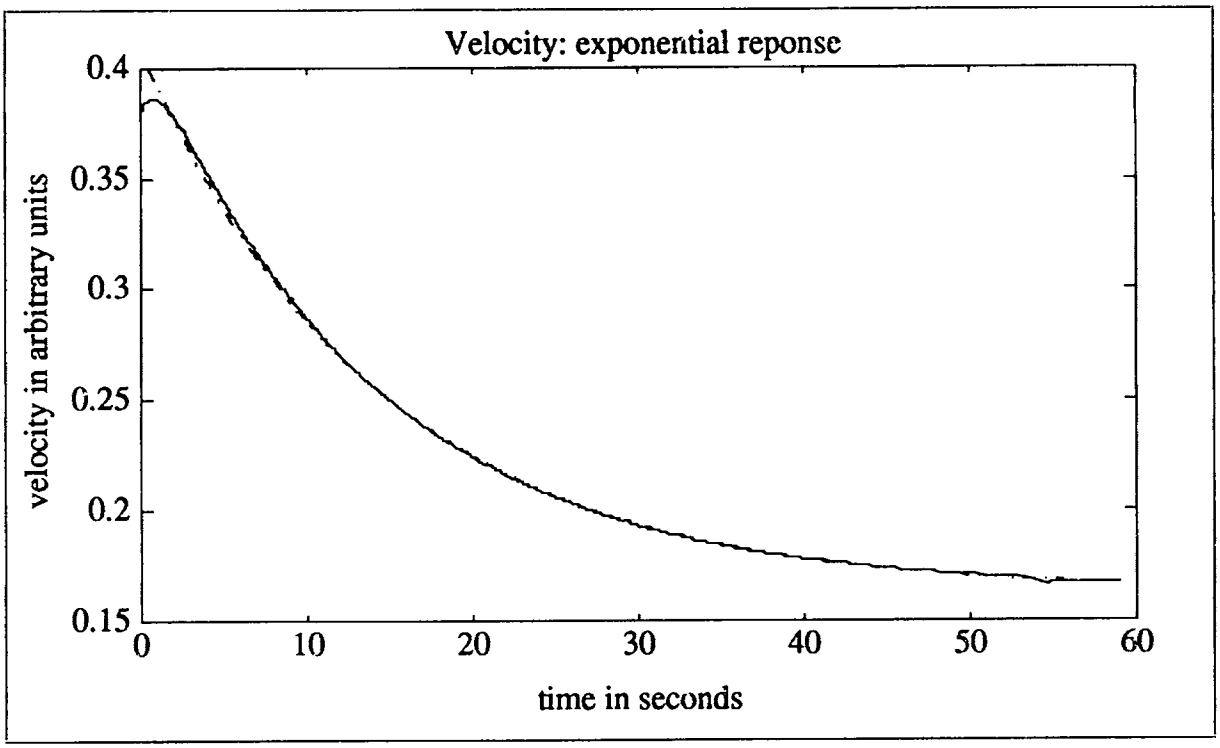
$$\omega = \frac{10}{T * 2 * \pi} = 0.155 \text{ Hz}$$

This calculated frequency is within 3% of the stimulus frequency[24] and shows compensatory eye movements for the sensed gravito-inertial force. The whole signal (solid line) can therefore be fit (exponential fit is dashed, complete fit is dotted) :



[24]These frequencies would be much closer if calculated over more than one run or over a longer data segment.

The model predicts exponential output of the form:



with an exponential decay fit of

$$f(t) = A * e^{(-t/T)} + C$$

where

T = 14.40 (samples)

A = 0.24 (arbitrary unist)

C = 0.16 (arbitrary unist)

The time constant of the exponentially decaying eye velocity is 14.40 seconds and longer than the actual data (cross-correlation coefficient between the fitted experimental data and the model prediction of 0.996).

However, it seems that the MEEI data run[25] is not very characteristic of normals. It is to be noted that other authors have published data resembling my model more than the MEEI data as shown in Table 9.

	MEEI run	MEEI average	Stockwell et al.	Model
Time constant	11.60 s	9.46 s	≈15 s	14.40 s
Amplitude	41.8°/s	24.7°/s	≈32°/s	24.00°/s
Bias	11.8°/s	13.2°/s	≈17°/s[26]	16°/s
Modulation	9.4°/s	5.7°/s	≈9°/s	11°/s

Table 9: Comparison of barbecue spit data.

Assuming a single semicircular canal for simplification, the model equations are:

$$\text{eq.1: } \dot{\vec{Y}} = \vec{\omega} - \frac{\vec{Y}}{\tau}$$

$$\text{eq.2: } \dot{\vec{g}} = \vec{\omega} \otimes \vec{g}$$

$$\text{eq.3: } \vec{e} = \overrightarrow{\text{error}}\left(\vec{g}, \hat{\vec{g}}\right)$$

$$\text{eq.4: } \dot{\hat{\vec{g}}} = \left(\hat{\vec{\omega}} + k_3 * \vec{e}\right) \otimes \hat{\vec{g}}$$

$$\text{eq.5: } \vec{a} = \vec{g} - \hat{\vec{g}}$$

$$\text{eq.6: } \dot{\hat{\vec{Y}}} = \hat{\vec{\omega}} - \frac{\hat{\vec{Y}}}{\hat{\tau}}$$

$$\text{eq.7: } \hat{\vec{\omega}} = k_1 * \left(\vec{Y} - \hat{\vec{Y}}\right) + k_2 * \vec{e}$$

As shown in Figure 1 and explained in Section III.2, eq.3 expresses that gravity error is calculated as the rotation required to align gravity with the expected gravity. Equations

[25] I could only obtain a single run from MEEI. At best, additional runs would probably have shifted the data toward the published MEEI averages for a normal. More likely, the data would have shifted towards the more widely accepted averages. In any case, this data run shows proof of concept and the system should be tested more thoroughly.

[26] [Benson and Bodin 1966] present a bias value of 15.4°/s, and [Correia and Guedry 1966] present a bias value of 14.1°/s.

eq.2 and eq.4 calculate the change in the gravity vector $\vec{g}$ and the estimated gravity vector $\hat{\vec{g}}$ respectively as the body rotates in space at a speed of $\vec{\omega}$ and $\hat{\vec{\omega}}$ respectively, using vector cross product. Additionaly, $\hat{\vec{g}}$ is compensated for directly perceived gravity error $\vec{e}$. Eq.7 calculates estimated velocity $\hat{\vec{\omega}}$ as the weighted sum of the semicircular canal error $\vec{Y} - \hat{\vec{Y}}$ and the gravity error $\vec{e}$.

The model contains two gravity vector calculations (eq.2 and eq.4), and one gravity error calculation (eq.3). These equations are translated into constraints using the non-linear extensions explained in Section IV.2.

The system requires modification of three parameters: k_1 has to be raised while both k_2 and k_3 have to be lowered. The model does indeed require changes characterizing the MEEI data as outside the normal range.

V.3.2. Hyper-Sensitive Canal-Otolith Interaction

Simulated data for hyper-sensitive canal-otolith interaction requires a significantly higher value for k_2. This model modification requires increasing the weight of the gravity error. The semicircular canal decay is still present but the velocity estimate remains almost constant, resembling data for monkeys [Merfeld 1990]. This response is shown in Figure 12 as a dotted line. The solid line represents the normal response for reference.

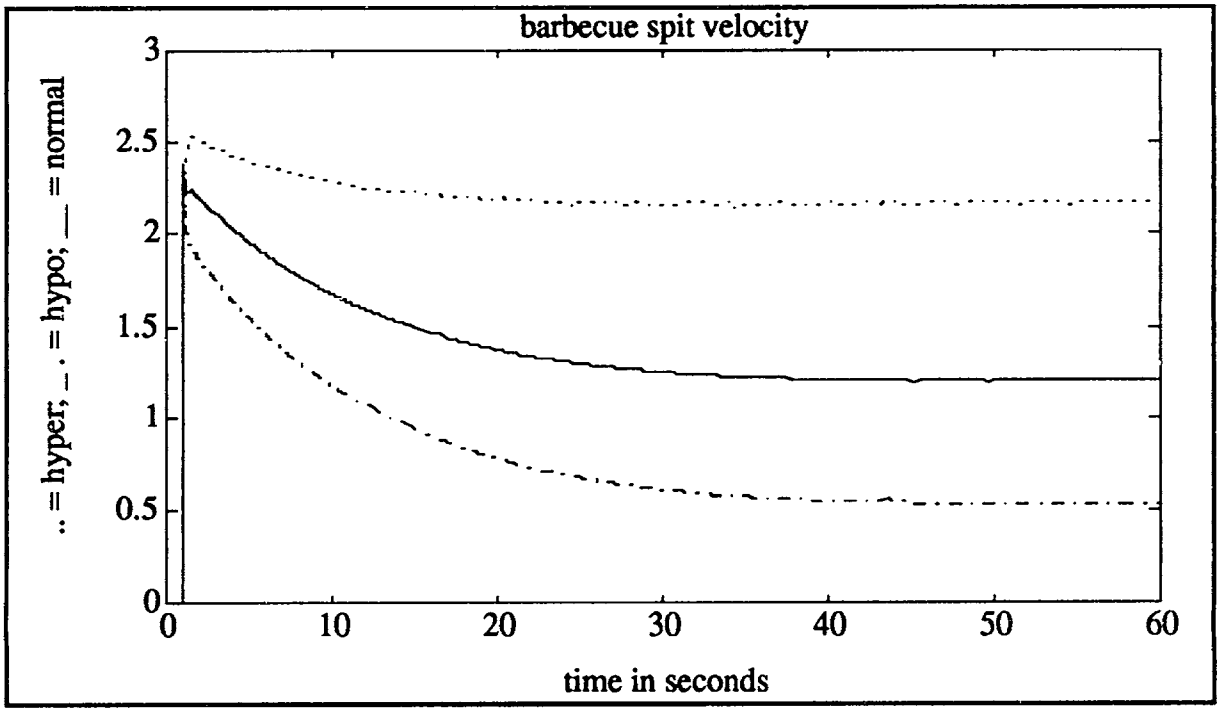


Figure 12: Barbecue spit simulated data.

MARIKA correctly suggest to modify k_2 out of the normal range to model hyper-sensitivity of the canal-otolith interaction.

V.3.3. Low Otolith Sensitivity

Simulated data for bilateral low sensitivity of the otolith does not provide the expected results. This points out a weakness of the model. If the subject recently suffered the condition, one would expect the otolith afferent signal to be low, while the expected otolith afferent would remain high. A large error would occur and drive the internal estimate low. The overall velocity response would show a decay to a lower final value. However, the gravity error calculation provided in the model is based on the angle

between the gravity vector and the expected gravity vector. This angle doesn't depend on the norm of the vectors and therefore is not sensitive to low otolith afference.

As a limit case, I have run the model assuming no otolith input. The response should then be identical to the chair stimulus. The model behaves well in this case and MARIKA tries to lower k_2. If the bounds on k_2 are tight, the model suggests a structural change in which the otoliths are taken out. If they are loose, the system proposes to lower k_2 to 0.1 which provides a response undistinguishable from the correct one. Interestingly, this is also the way the vestibular system would react. Seeing a suddenly high error with low signal to noise ratio from the otolith the vestibular system would decrease the otolith feedback gain k_2 and simultaneously decrease its internal representation of otolith sensitivity. As explained earlier, the model represents otolith error in a way that is not sensitive to the second change and therefore does not propose it.

This response is shown in Figure 12 as a dashed line. The solid line represents the normal response for reference.

CHAPTER 6

Conclusions

Nothing would get done if we succumbed to satisfaction. [Minsky 1986]

We would not need to deal with exceptions and censors if we lived in a universe of simple, general rules with no exceptions, as in the lovely mathematical worlds of arithmetic, geometry and logic. But perfect logic rarely works in the real world of people, thoughts, and things. This is because it is no accident that there are no exceptions to the rules in those mathematical worlds: *there, we start with the rules and imagine only objects that obey them.* But we can't so willfully make up the rules for objects that already exist, so our only course is to begin with imperfect guesses—collections of rough and ready rules—and then proceed to find out where they're wrong. [Minsky 1986]

VI. Conclusions

I presented a system that modifies the theory contained in a model of the normal human orientation system. Each parameter of the model is represented as a constraint on the range of possible values for normal subjects. As the examination of an individual progresses through a series of clinical tests, experimental signals are discretized. The structure of the model is extracted by generalizing the shape parameters with curve fit variables. The equations describing the model are translated into constraints on these variables. Two other types of constraints are also added to the network, initial values and differentiation constraints. Constraint propagation is then used to perform a qualitative simulation of the constraints stemming from normality and the various test results. If the individual is a normal subject, the system constrains the set of parameters describing the vestibular system of that individual. If the individual is a pathological case, the system is unable to find a consistent set of parameters. The constraints involved in the contradictory constraints directly suggest structural modifications of the system according to *a priori* knowledge of vestibular physiological failures. If numerical constraints are violated, the system relaxes them. The system then tunes the modified model. The final result is a model structurally different from that for a normal subject but that represents the specific subject's pathological vestibular system. This method is fundamentally different from a rule-based diagnostic system because it doesn't rely on patient symptoms and relations between symptoms and diseases. Instead, the system relies on detailed knowledge of a model of the patient and possible variations to the model. There is no direct indexing of these modifications by observables.

This system demonstrates automated scientific discovery in an actual scientific domain through techniques adapted from diagnosis and design. Qualitative and quantitative techniques are integrated in this single scheme, creating synergy between simulation and

AI techniques. The overall system represents progress in vestibular research, theory formation, and could prove useful for vestibular diagnosis.

VI.1. MARIKA's Development Context

The system I describe here is developed as part of a long term goal for an Interesting Data Filter software module. This module is one of the pieces of an intelligent computer system designed to assist astronauts while performing a space-borne vestibular experiment [Young et al. 1986, Young et al. 1989]. The computerized assistant is named [PI] (pronounced PI-in-a-box). This system is being developed jointly by the Man-Vehicle Laboratory at MIT and the AI Research Branch of the NASA Ames Research Center. The Aerospace Human Factors Research Division at Ames helps in designing the astronaut-computer interface. [PI] is being designed to fly on the SLS-2 Spacelab mission in Fall 1993. The system will be connected to the flight experiment apparatus and will assist the astronaut in the following tasks: Experiment Protocol Management, Data Acquisition and Reduction, Data Quality Monitoring, Diagnosis and Troubleshooting, Interesting Data Filtering, and Experiment Suggesting. In its long term version, this module would comprise individual vestibular models for each astronaut. These would be tuned according to the techniques developed here using baseline data obtained on the ground. The role of the system would be to compare flight data to expected data, flag discrepancies and provide adequate scientific explanations to the astronaut operator. On the basis of interaction, the system would then propose a new model. The Experiment Suggester module would be triggered after the astronaut accepted results from the Interesting Data Filter. Its role would be to modify the experiment to test the latest

hypothesis. Such a module could benefit from other work done in the scientific discovery field.

VI.2. Scientific Contributions

The MARIKA system is an inter-disciplinary piece of work that combines ideas from the AI, simulation, modeling, engineering design, and diagnosis domains. In return, it contributes to the fields of scientific discovery and vestibular research.

MARIKA demonstrates an automated discovery system on an actual scientific domain. Though the domain is clinical and not active vestibular research, I have paid attention to understanding the domain thoroughly and crafting a representation that makes reasoning natural and efficient. Extending the system to vestibular space adaptation should require mostly knowledge, not new paradigms.

MARIKA contributes to the vestibular domain through the modeling development it entailed. The model originally built from monkey data was extended to humans. I also added other senses such as vision and tactile cues. Modeling of separate symmetrical semicircular and otoliths end-organs provides a useful clinical research tool.

MARIKA demonstrates the use of techniques adapted from diagnosis and design that help solve a problem often believed to require some level of intuition. The system proves that the understanding of a mathematically modeled domain can be enhanced with the help of adapted AI techniques. MARIKA provides partial proof that:

> Because diagnosis can be described as a special case of abduction, it is [also] worthwhile to try exploiting computational mechanisms developed for model-based diagnosis to address more general abduction problems. [Hamscher 91]

Qualitative and quantitative analysis techniques are applied in synergy in a single scheme. Quantitative techniques are often more computationally expensive than qualitative ones. However, they often provide too much detail while qualitative techniques provide too few or produce too many under-constrained analyses. MARIKA shows that an appropriate mix of qualitative and quantitative representations and methods can be used in synergy to provide adequate detail at reasonable computational cost. Also, the system uses a discrete representation of time to reason about time-varying signals in an efficient manner.

MARIKA applies simulation and AI techniques in synergy.

> The growing cross-fertilization of ideas between the fields of artificial intelligence (AI) and simulation has stimulated interest in both the simulation and the artificial intelligence communities.
>
> ...
>
> Simulationists find that simulation software is weak: Users demand more realistic models, more support for the novice, and better assistance with extracting information from large amounts of data.
>
> ...
>
> Artificial intelligence workers find that their programs break down when dealing with real-world problems: They have difficulty in reasoning about time-varying phenomena and in predicting behavior of complex probabilistic systems. These limitations prevent expert systems from achieving their full potential in planning, diagnostic, advisory, and similar applications that require flexible and robust understanding of the behavior of realistically complex problems. [Widman and Loparo 1989]

Though answering only a few of these questions, MARIKA provides a more intelligent simulation system because of its AI component and a more realistic reasoner because of its simulation component.

Though limited to linear or steady state simulations, MARIKA is a step toward a useful clinical diagnostic tool. As is, the system is purposely lacking vestibular diagnostic

knowledge. It could be complemented to provide a diagnosis classification together with its altered model.

VI.3. Limitations of MARIKA

MARIKA works well with models that can be represented as linear differential equations. Completeness of the set of operators is easily satisfied within the domain of shapes. Indeed, the domain of shapes could very easily be extended by providing calculation methods for the various operators (basic operations, derivation, initial value). However, only a few non-linear cases are handled. All these cases yield shapes that would also be produced by linear operations. The transient state of the vestibular system cannot be studied with the system as it stands. Though most of the information can be read in the steady state model, some effects manifest themselves only as transients.

The system handles four basic shapes at this stage but could easily be extended to others. However, there is no general mathematical mechanism capable of recognizing or handling novel shapes. The discretization of data relies on specific knowledge of the shape, not generally applicable heuristics.

The segmentation of the signals is critical to proper approximation of the data. Vestibular experiments present well segmented inputs that provide all the information necessary to segment intermediary and output signals. Other domains could require segmentation algorithms as in the Flite system [Prager et al 1989].

MARIKA requires a working model of the domain. The current clinical domain satisfies this requirement when the normal human vestibular system is used. If the vestibular adaptation domain were used, an initial hypothetical model would be used. However,

many domains do not have established models. The goal of MARIKA is to refine a model, not create one from data.

MARIKA's model patching includes model parameter range adjustment and structural patching. The structural patching mechanism is very simple as it uses a table look-up. The quality of the patches depends on the foresight of the scientist who established the table. A knowledge-based patch suggester would enhance the opportunist attitude of the system.

MARIKA only suggest destructive model patches. It suggest malfunction or absence of sub-parts of the model which can be considered as degenerative. As outlined earlier, modifying the system for the vestibular adaptation domain would require constructive structural patches and possibly reorganization patches that would suggest new links between end-organs and signals.

MARIKA uses Screamer, an off-the-shelf constraint propagator that was slightly modified. Only a few of Screamer's capabilities are used by MARIKA, but computing penalties are paid for all of its overhead

VI.4. Future Work

MARIKA is currently limited to linear differential equations and consequently steady state only for some interesting and popular experimental conditions. It would be interesting to extend the model in general ways to include non-linear operators present in the vestibular model without restricting them to special simple cases.

In order to demonstrate the usefulness of model-based discovery as diagnosis and design, MARIKA should be applied to another domain. This domain could involve a different set

of shapes. The domain would have to be a linear one unless further work lifts that restriction or special cases can be found to avoid it.

It would be interesting to modify MARIKA to provide its own set of model alterations. Because MARIKA uses object representation for all the mathematical objects it manipulates, this could be done in a way similar to AM by applying mutation operators to the content of object slots. Domain heuristics would have to be applied to generate meaningful mutations. For expression objects, these patches would be indexed by the original form of the expression. The variable patches would still be more efficiently implemented as the set of heuristics present in MARIKA. In a system modified along these lines, incremental patch generation and indexing would take place after every use of a patch to allow further modification.

To be most useful in a clinical setting, MARIKA would have to be modified to help find the next test that would best discriminate between competing structurally different models. The system would require knowledge of the available experimental setups and evaluate predictions of the competing models and choose the test that maximizes the information gained on the patients vestibular system. Such a system would greatly improve the time necessary for diagnostic testing by shortening the test-analysis loop which is often a limiting factor in a clinical setting. To achieve these capabilities, the current system must first be rendered seamless and its speed must be improved. Another module must be developed to compare predictions made by competing models and suggest tests that would differentiate them. This new module could also encode knowledge on the cost of the test and perform some optimization on its test selection.

An important next step in model-based scientific discovery is to modify MARIKA to solve the original problem of theory formation within the domain of space vestibular

adaptation. Once this step is achieved, making the system perform under space flight time and hardware performance requirements would allow it to be tested as a true scientific discovery system.

References

Apple Computer, Inc., *Macintosh Common Lisp 2.0 Reference*, 030-5008-A, Developer Technical Publications, 1991.

Balkwill M. D., *Changes in Human Horizontal Angular VOR After the Spacelab SLS-1 Mission*, Unpublished Master's thesis, Aeronautics and Astronautics Department, Massachusetts Institute of Technology, February 1992.

Benson A. J., and Bodin M. A., *Interaction of Linear and Angular Accelerations on Vestibular Receptors*, in Manual of Aerospace Medicine, 37, 144-154, 1966.

Bohr N., *Atomic Theory and the Description of Nature*, 1929.

Changeux J.-P. *L'Homme Neuronal*, Librairie Arthème Fayard, 1983.

Correia M. J., and Guedry F. E., *Modification of Vestibular Responses as a Function of Rate of Rotation About an Earth-Horizontal Axis*, Acta Otolaryngologica, 62, 297-308, 1966.

Eykhoff P., *System Identification*, Wiley, New York, 1974.

Falkenheimer B. C., and Michalski R. S., *Integrating Quantitative and Qualitative Discovery: The ABACUS System*, Machine Learning, 1: 367-401, 1986.

Forbus K. D., *Interpreting Observations of Physical Systems*, IEEE Transactions on Systems, Man, and Cybernetics, SMS-17 (3), 350-359, 1987.

Forbus K. D., *Qualitative Process Theory*, Artificial Intelligence. 24, 85-168, 1984.

Furman J. M. R., *Earth Horizontal Axis Rotational Responses in Patients with Unilateral Peripheral Vestibular Deficits*, Annals of Otology, Rhinology and Laryngology, Vol. 98, No. 7, 551-555, July 1989.

Furman J. M. R., Wall C. III, and Pang D., *Vestibular Function in Periodic Alternating Nystagmus*, Brain, 113: 1425-1439, 1990.

Groleau N., *A Blackboard Architecture for Communication and Coordination in Design*, unpublished Master's Thesis, Dept. of Civil Engineering, Massachusetts Institute of Technology, February 1989.

Hamscher W, and Davis R., *Issues in Model Based Troubleshooting*, A.I. memo 893, Massachusetts Institute of Technology, March 1987.

Hamscher W., *Principles of Diagnosis: Current Trends and a Report of the First International Workshop*, AI Magazine, 15-23, Winter 1991.

Hawking S. W., *A Brief History of Time, From the Big Bang to Black Holes*, Bantam Books, April 1988.

Imagine That, Inc., *Extend™: Generic and Discrete Event Libraries*, February 1990.

Kalman R. E., and Bucy R. S., *New Results in Linear Filtering and Prediction Theory*, Journal of Basic Engineering, Transactions of the ASME, Ser D 83:95-108, 1961.

Karp P. D., and Friedland P., *Coordinating the Use of Qualitative and Quantitative Knowledge in Declarative Device Modeling*, in Artificial Intelligence, Modeling and Simulation, Wiley, New York, 189-206, 1989.

Karp P. D., *Hypothesis Formation and Qualitative Reasoning in Molecular Biology*, unpublished Ph.D. Thesis, STAN-CS-89-1263, Stanford University, 1989.

Kocabas S., *Elements of Scientific Research: Modeling Discoveries in Oxide Superconductivity*, in Proceedings of the Workshop on Machine Discovery, July 1992, Aberdeen, Scotland.

Kokar M. M., *Determining Arguments of Invariant Functional Descriptions*, Machine Learning, 1: 403-422, 1974.

Kulkarni D. S., and Simon H., *The Processes of Scientific Research: The Strategy of Experimentation*, Cognitive Science, 12:139-175, 1988.

Kulkarni D. S., *The Processes of Scientific Research: The Strategy of Experimentation*, unpublished Ph.D. Thesis, Carnegie Mellon University, 1987.

Langley P., and Nordhausen B., *A Framework for Empirical Discovery*, in Proceedings of the International Meeting on Advances in Learning, Les Arcs, France, 1986.

Langley P., and Zytkow J. M., *Data-Driven Approaches to Empirical Discovery*, Artificial Intelligence, 40, 283-312, 1989.

Langley P., Simon H. L., Bradshaw G. L., and Zytkow J. M., *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, Cambridge, 1987.

Lenat D. B., and Brown J. S., *Why AM and EURISKO Appear to Work*, Artificial Intelligence, 23, 269-294, 1984.

Lenat D. B., *Automated Theory Formation in Mathematics*, in Proceedings IJCAI-77, 833-842, 1977.

Merfeld D. M., *Spatial orientation in the squirrel monkey: An Experimental and Theoretical Investigation*, unpublished Ph.D. Thesis, Massachusetts Institute of Technology, February 1990.

Merfeld D. M., Young L. R., Oman C. M., and Shelhamer M. J., *A Multi-dimensional Model of the Effect of Gravity on the Spatial Orientation of the Monkey*, submitted to the Journal of Vestibular Research, May 1992.

Minsky M., *The Society of Mind*, Simon and Schuster, 1986.

Mira E., Schmid R., Zanocco P., Buizza A., Magenes G., and Manfrin M., *A Computer-Based Consultation System (Expert System) for the Classification and Diagnosis of Dizziness*, Adv. Oto-Rhino-Laryngology, vol. 42., pp. 77-80, Karger, Basel, 1988.

Morrison P., and Morrison P., *The Ring of Truth, An Inquiry in How We Know What We Know*, Random House, 1987.

NASA, *Spacelab Life Science 1, Reprints of Background Life Sciences Publications*, White R., and Leonard J. eds., 1991.

Newell A., and Simon H. A., *GPS, a program that simulates human thought*, in Computers and Thought, Feigenbaum and Feldman eds., McGraw-Hill, New York, 1963.

Oman C. M., *Sensory Conflict in Motion Sickness: an Observer Theory Approach*, Symposium and Workshop on Spatial Displays and Spatial Instruments, Alisomar, California, September 1988.

Ormsby C. C., and Young L. R., *Integration of semicircular canal and otolith information for multisensory orientation stimuli*, Mathematical Biosciences 34:1-21, 1977.

Piatetsky-Shapiro G., and Frawley W., *Proceedings of the International Joint Conferences on Artificial Intelligence. Workshop on Knowledge Discovery in Databases*, Menlo Park, CA, 1989.

Prager R., Belanger P., and De Mori R., *A Knowledge-Based System for Troubleshooting Real-Time Models*, in Artificial Intelligence Simulation & Modeling, L. E. Widman, K. A. Loparo, and N. R. Nielsen, eds., Wiley, New York, 1989.

Raphan T., Matsuo V., and Cohen B., *A velocity storage mechanism responsible for optokinetic nystagmus (OKN), optokinetic after-nystagmus (OKAN) and vestibular nystagmus, Control of Gaze by Brain Neurons*, in Developments in Neuroscience, Vol. 1., Elsevier/North Holland Biomedical Press, 37-47, 1977.

Robinson D. A., *Vestibular and optokinetic symbiosis: an example of explaining by modeling, Control of Gaze by Brain Neurons*, in Developments in Neuroscience, Vol. 1., Elsevier/North Holland Biomedical Press, 49-58, 1977.

Rosenblueth A., and Wiener N., *The Role of Models in Science*, IEEE Transactions on Automatic Control, AC-14, 270-276, 1945.

Schement M. J., and Hartline H. P., *An Intelligent Controller for Neurophysiological Experiments*, in Computer-Based Medical Systems, Proceedings of the Fifth Annual IEEE Symposium, Durham, North Carolina, 528-538, June 1992.

Shell P., and Carbonell J., *PARMENIDES: A Class-Based Frame System*, Carnegie Mellon University, January 1990.

Simmons R. G., *Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems*, unpublished Ph.D. Thesis, Massachusetts Institute of Technology, 1988.

Siskind J. M., *Screaming Yellow Zonkers*, Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Document, 1991.

Sriram D., and Tong C., *Artificial Intelligence and Engineering Design*, Ninth National Conference on Artificial Intelligence, Tutorial SA2, Anaheim, CA, July 1991.

Stockwell C. W., Turnispeed G. T., and Guedry F. E., *Nystagmus Responses During Rotation about a Tilted Axis*, Army-Navy Joint Report, no. NAMRL-1129, Pensacola: Nav Aerospace Med Res Lab, 1971.

The Math Works, Inc., *MATLAB™ User's Guide*, 1989.

The Math Works, Inc., *SIMULAB™ A Program for Simulating Dynamic Systems, User's Guide*, 1991.

Valdés-Pérez R. E., *Machine Discovery of Chemical Reaction Pathways*, Ph.D. Dissertation, CMU-CS-90-191, School of Computer Science, Carnegie Mellon University, 1990.

Wall C. III, and Black F. O., *Intersubject Variability in VOR Responses to 0.005—1.0 Hz Sinusoidal Rotations*, Acta Otolaryngologica, 406: 194-198, 1984.

Wall C. III, and Furman J. M. R., *Nystagmus Responses in a Group of Normal Humans during Earth-horizontal Axis Rotation*, Acta Otolaryngologica, 108: 327-335, 1989.

Wall C. III, Black F. O., and Hunt A. E., *Effects of Age, Sex and Stimulus Parameters upon Vestibulo-ocular Responses to Sinusoidal Rotations*, Acta Otolaryngologica, 406: 194-198, 1984.

Wall C. III, Busis S., and Kamerer D. B., *Differential Assessment of Otolithic Versus Canal Function in Patients Using Earth Horizontal Axis Rotation*, in The Vestibular Neurophysiologic and Clinical Research, Graham, Malcom D., and Kemink, John L., eds., Raven Press, New York, 263-270, 1987.

Wall C. III, *The Sinusoidal Harmonic Acceleration Rotatory Chair Test—Theoretical and Clinical Basis*, Neurologic Clinics, Vol. 8, No. 2, 269-285, May 1990.

Widman L. E., and Loparo K. A., *Artificial Intelligence, Simulation and Modeling: A critical Survey*, in Artificial Intelligence Simulation & Modeling, L. E. Widman, K. A. Loparo, and N. R. Nielsen, eds., Wiley, New York, 1989.

Young L. R., Colombano S. P., Haymann-Haber G., Groleau N., Szolovits P., and Rosenthal D., *An Expert System to Advise Astronauts During Experiments*, In the Proceedings of the International Astronautical Congress, Malaga, Spain, 1989.

Young L. R., *Perception of the body in space: mechanisms*, Handbook of Physiology, The Nervous System III, Chapter 22, American Physiological Society, Smith I. D., ed., 1984.

Young L. R., Shelhamer M., and Modestino S., *MIT/Canadian Vestibular Experiments on the Spacelab-1 mission: 2. Visual Vestibular Tilt Interaction in Weightlessness*, in Experimental Brain Research, 64: 299-307, 1986.

Zytkow J. M., and Baker J., *Interactive Mining of Regularities in Databases*, in Knowledge Discovery in Databases, Piatetsky-Shapiro G., and Frawley W. eds., AAAI press, 1991.

Zytkow J. M., *Combining Many Searches in the FAHRENHEIT Discovery System*, in Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, 281-287, 1987.

APPENDIX A

MATLAB Curve Fitting Code

MATLAB/SIMULAB for the Macintosh version 1.1

This code allows the fitting of any type of multi-parameter function. Constrained optimization is done according to Mean Squared Error with a call to `constr`, a Nelder-Meade search algorithm of the parameters.

```
function [e,g] = model_err_sk(model_parms,t,u,f,norm_parms)

%model_err_nick
% Error function for model fitting. Constrained optimization
% minimizes the output of this function, which is currently set
% as the mean square error (MSE) between the data and the
% fitting function.
% T. Liefeld 12/6/92
% Modified N. Groleau 7/92
% calculate physical parameters for transfer function, and
% determine the corresponding model response

model_parms = model_parms * norm_parms;
T = model_parms(1);
A = model_parms(2);
P = model_parms(3);
C = model_parms(4);
K = model_parms(5);

% Define only one function to fit to
y=A*exp(-1*t/T)+C;
%y=A*(1- exp(-1*t/T))+C*(1- exp(-1*t/P))+K;
%y=A*sin(t/T + P)+C;

% ensure that y and f are both either row vectors or column vectors
[m2,n2] = size(f);
[m1,n1] = size(y);
if (m1 > n1)            % y is column vector
        if (m2 < n2)    % f is row vector
        y = y';
        end
else                    % y is row vector
        if (m2 > n2)    % f is column vector
        y = y';
        end
end

% Only base MSE on data points at which we have valid data.

d=abs(y-f);
e=sum(d * d) / length(d);

% Un-comment the following four lines for plotting during fitting
%plot(t,f)
%hold on
%plot(t,y,'g');
%hold off

% dummy value which 'constr' requires but is unused for my
% purposes; this must be some constant value for my purposes

g = -1;

return;
```

```
%nick_model_fit
% D. Balkwill 12/9/91
% modified T. Liefeld 06/12/92,7/2/92
% for a different function and parameters.
% modified N. Groleau 7/92 for various functions
% and additional parameters
% Fits a model of up to 5 parameters
% Fitting function calculated as y(t) in model_err_nick
% Curve to fit is f(t) defined in the environment

global y;
u=ones(t);

% Nominal model parameters. The parameters to be fitted are the
% non-dimensional ratios of the physical parameters to the
% nominal model parameters here. This places equal emphasis
% on each model parameter, even though they may be orders of
% magnitude apart.

T = 17.5;       % time constant
A = 0.4;        % amplitude
P = 0.0;        % phase
C = 0.0;        % additive constant
K = 0.0;        % additiveconstant

norm_parms = [T; A; P; C; K];

%first option is 1 for verbose, 0 for silent
%error tolerances -- see "help foptions"
options = [1 ; 1e-7 ; 1e-7; 1e-7; 1e-7; 1e-7];
vlb = [0.8; 0.8; 0.8; 0.8; 0.8]; %lower bounds
vub = [1.2; 1.2; 1.2; 1.2; 1.2]; %upper bounds

model_parms = [1; 1; 1; 1; 1];
[model_parms, options] = constr('model_err_nick', model_parms, options, vlb, vub, [], t,
u, f, norm_parms);

model_parms = model_parms * norm_parms;

fprintf('*** Model fit ***');
fprintf('Number of iterations = %5.0f\n',options(10));
fprintf('Mean square error = %7.5f\n',options(8));

fprintf('T = %f\n',model_parms(1));
fprintf('A = %f\n',model_parms(2));
fprintf('P = %f\n',model_parms(3));
fprintf('C = %f\n',model_parms(4));
fprintf('K = %f\n',model_parms(5));

% plotting the original function and the fit
plot(t,f);
hold on;
plot(t,y,'g');
hold off;
```

APPENDIX B

SCREAMER Patch Lisp Code

This code provides useful names for the constrained variables generated by Screamer. It also provides a message trace indicating which constrained failed.

```
;;; trace.lisp
;;; N. Groleau 6/92
;;; Most of this code is verbatim copy of Screamer. Comments with my modifications.

(in-package :screamer)

(use-package '(:lisp))

(lisp:defun fail () (throw 'fail nil))

;;;the OPv2 definitions give meaningful names to the internally generated variables
;;; implements (+v x y)
(defun +v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((and (not (variable? x)) (zerop x)) y)
        ((and (not (variable? y)) (zerop y)) x)
        ((and (not (variable? x)) (not (variable? y))) (+ x y))
        (t (let ((z (make-variable
                     `(+ ,(if (variable? x) (variable-name x) x)      ;;; variable is called
                         ,(if (variable? y) (variable-name y) y)))))  ;;; (+ name-of-x name-of-y)
             (assert!-numberv z)
             (+-rule z x y)
             (setf x (value-of x))
             (setf y (value-of y))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (+-rule z x y)) x)
             (attach-noticer! #'(lambda () (+-rule z x y)) y)
             (attach-noticer! #'(lambda () (+-rule z x y)) z)
             z)))))

;;; implements (-v x y)
(defun -v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((and (not (variable? x)) (zerop x)) (- y))
        ((and (not (variable? y)) (zerop y)) x)
        ((and (not (variable? x)) (not (variable? y))) (- x y))
        (t (let ((z (make-variable
                     `(- ,(if (variable? x) (variable-name x) x)      ;;; variable is called
                         ,(if (variable? y) (variable-name y) y)))))  ;;; (- name-of-x name-of-y)
             (assert!-numberv z)
             (+-rule x y z)
             (setf x (value-of x))
             (setf y (value-of y))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (+-rule x y z)) x)
             (attach-noticer! #'(lambda () (+-rule x y z)) y)
             (attach-noticer! #'(lambda () (+-rule x y z)) z)
             z)))))
```

```
;;; implements (*v x y)
(defun *v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((and (not (variable? x)) (zerop x)) 0)
        ((and (not (variable? y)) (zerop y)) 0)
        ((and (not (variable? x)) (= x 1)) y)
        ((and (not (variable? y)) (= y 1)) x)
        ((and (not (variable? x)) (not (variable? y))) (* x y))
        (t (let ((z (make-variable
              `(* ,(if (variable? x) (variable-name x) x)        ;;; variable is called
                  ,(if (variable? y) (variable-name y) y)))))    ;;; (* name-of-x name-of-y)
           (assert!-numberv z)
           (*-rule z x y)
           (setf x (value-of x))
           (setf y (value-of y))
           (setf z (value-of z))
           (attach-noticer! #'(lambda () (*-rule z x y)) x)
           (attach-noticer! #'(lambda () (*-rule z x y)) y)
           (attach-noticer! #'(lambda () (*-rule z x y)) z)
           z)))))

;;; implements (/v x y)
(defun /v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((and (not (variable? x)) (zerop x)) 0)
        ((and (not (variable? y)) (zerop y)) (fail))
        ((and (not (variable? y)) (= y 1)) x)
        ((and (not (variable? x)) (not (variable? y))) (/ x y))
        (t (let ((z (make-variable
              `(/ ,(if (variable? x) (variable-name x) x)        ;;; variable is called
                  ,(if (variable? y) (variable-name y) y)))))    ;;; (/ name-of-x name-of-y)
           (assert!-numberv z)
           (*-rule x y z)
           (setf x (value-of x))
           (setf y (value-of y))
           (setf z (value-of z))
           (attach-noticer! #'(lambda () (*-rule x y z)) x)
           (attach-noticer! #'(lambda () (*-rule x y z)) y)
           (attach-noticer! #'(lambda () (*-rule x y z)) z)
           z)))))
```

```
;;; implements (=v x y)
(defun =v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((known?-=v2 x y) t)
        ((known?-/=v2 x y) nil)
        (t (let ((z (make-variable
              `(= ,(if (variable? x) (variable-name x) x)        ;;; variable is called
                  ,(if (variable? y) (variable-name y) y)))))    ;;; (= name-of-x name-of-y)
             (assert!-booleanv z)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-=v2 x y) (assert!-true z))
                        ((known?-/=v2 x y) (assert!-false z))))
              x)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-=v2 x y) (assert!-true z))
                        ((known?-/=v2 x y) (assert!-false z))))
              y)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-true z) (assert!-=v2 x y))
                        ((known?-false z) (assert!-/=v2 x y))))
              z)
             z)))))
```

```
;;; implements (<=v x y)
(defun <=v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((known?-<=v2 x y) t)
        ((known?-<v2 y x) nil)
        (t (let ((z (make-variable
              `(<= ,(if (variable? x) (variable-name x) x)      ;;; variable is called
                   ,(if (variable? y) (variable-name y) y)))))  ;;; (<= name-of-x name-of-y)
             (assert!-booleanv z)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-<=v2 x y) (assert!-true z))
                        ((known?-<v2 y x) (assert!-false z))))
              x)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-<=v2 x y) (assert!-true z))
                        ((known?-<v2 y x) (assert!-false z))))
              y)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-true z) (assert!-<=v2 x y))
                        ((known?-false z) (assert!-<v2 y x))))
              z)
             z)))))
```

```
;;; implements (<v x y)
(defun <v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((known?-<v2 x y) t)
        ((known?-<=v2 y x) nil)
        (t (let ((z (make-variable
              `(< ,(if (variable? x) (variable-name x) x)        ;;; variable is called
                  ,(if (variable? y) (variable-name y) y)))))    ;;; (< name-of-x name-of-y)
             (assert!-booleanv z)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-<v2 x y) (assert!-true z))
                        ((known?-<=v2 y x) (assert!-false z))))
              x)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-<v2 x y) (assert!-true z))
                        ((known?-<=v2 y x) (assert!-false z))))
              y)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-true z) (assert!-<v2 x y))
                        ((known?-false z) (assert!-<=v2 y x))))
              z)
             z)))))
```

```
;;; implements (/=v x y)
(defun /=v2 (x y)
 (assert!-numberv x)
 (assert!-numberv y)
 (let ((x (value-of x))
       (y (value-of y)))
  (cond ((known?-/=v2 x y) t)
        ((known?-=v2 x y) nil)
        (t (let ((z (make-variable
              `(/= ,(if (variable? x) (variable-name x) x)      ;;; variable is called
                   ,(if (variable? y) (variable-name y) y)))))  ;;; (/= name-of-x name-of-y)
             (assert!-booleanv z)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-/=v2 x y) (assert!-true z))
                        ((known?-=v2 x y) (assert!-false z))))
              x)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-/=v2 x y) (assert!-true z))
                        ((known?-=v2 x y) (assert!-false z))))
              y)
             (attach-noticer!
              #'(lambda ()
                  (cond ((known?-true z) (assert!-/=v2 x y))
                        ((known?-false z) (assert!-=v2 x y))))
              z)
             z)))))
```

```
;;; prints only reasonable precision
(defun print-variable (x stream print-level)
 (declare (ignore print-level))
 (let ((x (value-of x)))
  (if (variable? x)
     (cond ((and (not (eq (variable-domain x) t))
                 (not (null (variable-antidomain x))))
            (error "This shouldn't happen"))
           ((not (eq (variable-domain x) t))
            (format stream "[~S domain:~S]"
                    (variable-name x)
                    (variable-domain x)))
           ((not (null (variable-antidomain x)))
            (format stream "[~S antidomain:~S]"
                    (variable-name x)
                    (variable-antidomain x)))
           ((known?-realv x)
            (let ((type (cond
                         ((known?-integerv x) "integer")
                         ((known?-not-integerv x) "non-integer real")
                         (t "real"))))
             (if (finite-range-min? x)
                 (if (finite-range-max? x)
                     (format stream "[~S ~A ~4,2F:~4,2F]"   ;;; changed format to 4,2F
                             (variable-name x)
                             type
                             (range-min x)
                             (range-max x))
                     (format stream "[~S ~A ~4,2F:]"        ;;; changed format to 4,2F
                             (variable-name x)
                             type
                             (range-min x)))
                 (if (finite-range-max? x)
                     (format stream "[~S ~A :~4,2F]"        ;;; changed format to 4,2F
                             (variable-name x)
                             type
                             (range-max x))
                     (format stream "[~S ~A]"
                             (variable-name x)
                             type)))))
           ((known?-numberv x)
            (format stream "[~S ~A]"
                    (variable-name x)
                    (if (known?-not-realv x) "non-real number" "number")))
           ((known?-not-numberv x)
            (format stream "[~S non-number]" (variable-name x)))
           (t (format stream "[~S]" (variable-name x))))
     (format stream "~4,2F" x))))                          ;;; changed format to 4,2F
```

```
;;;tells me what minimum it failed on
(defun restrict-min! (x new-min)
 (assert!-realv x)
 (let ((x (value-of x))
       (run? nil))
  (cond ((variable? x)
         (when (or (not (finite-range-min? x)) (> new-min (range-min x)))
          ;; note: Should be fuzzy<.
          (if (and (finite-range-max? x) (< (range-max x) new-min))
         (progn
            (format t "~%Failed when attempting to set the min of ~A to ~4,2F" (variable-
name x) new-min)              ;; added warning message
          (fail)
         ))
          (local (setf (variable-min x) new-min))
          (setf run? t))
          (cond ((and (not (eq (variable-domain x) t))
                      (some #'(lambda (element) (known?-<v2 element new-min))
                            (variable-domain x)))
                 ;; note: Could do less consing if had LOCAL DELETE-IF.
                 ;;       This would also allow checking list only once.
                 (assign! x (remove-if #'(lambda (element)
                                           (known?-<v2 element new-min))
                                       (variable-domain x))))
              (run? (run-noticers x))))
         ;; note: Should be fuzzy<.
         (t (if (< x new-min)
          (progn
           (format t "~%Failed when attempting to set ~4,2F to ~4,2F" x new-min)
                 ;; added warning message
           (fail))
           )))))
```

```
;;;tells me what maximum it failed on
(defun restrict-max! (x new-max)
 (assert!-realv x)
 (let ((x (value-of x))
       (run? nil))
  (cond ((variable? x)
         (when (or (not (finite-range-max? x)) (< new-max (range-max x)))
          ;; note: Should be fuzzy>.
          (if (and (finite-range-min? x) (> (range-min x) new-max))
         (progn
            (format t "~%Failed when attempting to set the max of ~A to ~4,2F" (variable-
name x) new-max)              ;; added warning message
          (fail)
          ))
          (local (setf (variable-max x) new-max))
          (setf run? t))
         (cond ((and (not (eq (variable-domain x) t))
                     (some #'(lambda (element) (known?-<v2 new-max element))
                           (variable-domain x)))
                ;; note: Could do less consing if had LOCAL DELETE-IF.
                ;;       This would also allow checking list only once.
                (assign! x (remove-if #'(lambda (element)
                                          (known?-<v2 new-max element))
                                      (variable-domain x))))
               (run? (run-noticers x))))
        ;; note: Should be fuzzy>.
        (t (if (> x new-max)
         (progn
          (format t "%Failed when attempting to set ~4,2F to ~4,2F" x new-max)
        ;; added warning message
          (fail))
          )))))
```

```
;;; trigo.lisp
;;; N. Groleau 7/92
;;; Implements constraint for sinusoidal signals. Modeled after Screamer code.

(in-package :screamer)

(use-package '(:lisp))

(shadow '(defun))

(proclaim '(declaration magic))

(export '(cosv sinv tanv sqrtv))

;;; COS
(defun cosv-internal (xs)
 (if (null xs) 0 (cosv2 (first xs))))

(defun cosv (&rest xs) (cosv-internal xs))

(defun cosv2 (x)
 (assert!-numberv x)
 (let ((x (value-of x)))
  (cond ((not (variable? x)) (cos x))
        (t (let ((z (make-variable
                     `(cos ,(if (variable? x) (variable-name x) x))
                     )))
             (assert!-numberv z)
             (cos-rule z x)
             (setf x (value-of x))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (cos-rule z x)) x)
             (attach-noticer! #'(lambda () (cos-rule z x)) z)
             z)))))

(defun cos-rule (z x)
  (let ((x (value-of x))
        (z (value-of z)))
    (if (and (numberp z) (numberp x) (/= z (cos x))) (fail))
    (if (known?-realv x) (assert!-realv z))
    (when (known?-realv z)
      (assert!-realv x))
    (if (known?-not-realv x)
      (assert!-not-realv z))
    (when (and (known?-realv x) (known?-realv z))
   (if (finite-range-min? x)
      (restrict-max! z (cos (range-min x))))
   (if (finite-range-max? x)
      (restrict-min! z (cos (range-max x))))
   (if (finite-range-min? z)
      (restrict-max! x (acos (range-min z))))
   (if (finite-range-max? z)
      (restrict-min! x (acos (range-max z))))
   )))
```

```
;;; SIN

(defun sinv-internal (xs)
 (if (null xs) 0 (sinv2 (first xs))))

(defun sinv (&rest xs) (sinv-internal xs))

(defun sinv2 (x)
 (assert!-numberv x)
 (let ((x (value-of x)))
  (cond ((not (variable? x)) (sin x))
        (t (let ((z (make-variable
                     `(sin ,(if (variable? x) (variable-name x) x))
                     )))
             (assert!-numberv z)
             (sin-rule z x)
             (setf x (value-of x))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (sin-rule z x)) x)
             (attach-noticer! #'(lambda () (sin-rule z x)) z)
             z)))))

(defun sin-rule (z x)
  (let ((x (value-of x))
        (z (value-of z)))
    (if (and (numberp z) (numberp x) (/= z (sin x))) (fail))
    (if (known?-realv x) (assert!-realv z))
    (when (known?-realv z)
      (assert!-realv x))
    (if (known?-not-realv x)
      (assert!-not-realv z))
    (when (and (known?-realv x) (known?-realv z))
   (if (finite-range-min? x)
      (restrict-min! z (sin (range-min x))))
   (if (finite-range-max? x)
      (restrict-max! z (sin (range-max x))))
   (if (finite-range-min? z)
      (restrict-min! x (asin (range-min z))))
   (if (finite-range-max? z)
      (restrict-max! x (asin (range-max z))))
   )))
```

```
;;; TAN

(defun tanv-internal (xs)
 (if (null xs) 0 (tanv2 (first xs))))

(defun tanv (&rest xs) (tanv-internal xs))

(defun tanv2 (x)
 (assert!-numberv x)
 (let ((x (value-of x)))
  (cond ((not (variable? x)) (tan x))
        (t (let ((z (make-variable
                     `(tan ,(if (variable? x) (variable-name x) x))
                     )))
             (assert!-numberv z)
             (tan-rule z x)
             (setf x (value-of x))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (tan-rule z x)) x)
             (attach-noticer! #'(lambda () (tan-rule z x)) z)
             z)))))

(defun tan-rule (z x)
  (let ((x (value-of x))
        (z (value-of z)))
    (if (and (numberp z) (numberp x) (/= z (tan x))) (fail))
    (if (known?-realv x) (assert!-realv z))
    (when (known?-realv z)
      (assert!-realv x))
    (if (known?-not-realv x)
      (assert!-not-realv z))
    (when (and (known?-realv x) (known?-realv z))
    (if (finite-range-min? x)
        (restrict-min! z (tan (range-min x))))
    (if (finite-range-max? x)
        (restrict-max! z (tan (range-max x))))
    (if (finite-range-min? z)
        (restrict-min! x (atan (range-min z))))
    (if (finite-range-max? z)
        (restrict-max! x (atan (range-max z))))
    )))
```

```
;;; SQRT

(defun sqrtv-internal (xs)
 (if (null xs) 0 (sqrtv2 (first xs))))

(defun sqrtv (&rest xs) (sqrtv-internal xs))

(defun sqrtv2 (x)
 (assert!-numberv x)
 (let ((x (value-of x)))
  (cond ((not (variable? x)) (sqrt x))
        (t (let ((z (make-variable
                     `(sqrt ,(if (variable? x) (variable-name x) x))
                     )))
             (assert!-numberv z)
             (sqrt-rule z x)
             (setf x (value-of x))
             (setf z (value-of z))
             (attach-noticer! #'(lambda () (sqrt-rule z x)) x)
             (attach-noticer! #'(lambda () (sqrt-rule z x)) z)
             z)))))

(defun sqrt-rule (z x)
 (let ((x (value-of x))
       (z (value-of z)))
   (if (and (numberp z) (numberp x) (/= z (sqrt x))) (fail))
   (when (known?-integerv z)
     (assert!-integerv x))
   (if (known?-not-integerv x)
     (assert!-not-integerv z))
   (if (known?-realv x) (assert!-realv z))
   (when (known?-realv z)
     (assert!-realv x))
   (if (known?-not-realv x)
     (assert!-not-realv z))
   (when (and (known?-realv x) (known?-realv z))
   (if (finite-range-min? x)
       (restrict-min! z (sqrt (range-min x))))
   (if (finite-range-max? x)
       (restrict-max! z (sqrt (range-max x))))
   (if (finite-range-min? z)
       (restrict-min! x (* (range-min z) (range-min z))))
   (if (finite-range-max? z)
       (restrict-max! x (* (range-max z) (range-max z))))
   )))
```

APPENDIX C

PARMENIDES Object-Oriented Programming Lisp Code

This code provides a more flexible interface to Parmenides objects as well as implement a simple message passing scheme using the existing slot structure.

```
;;; signals.lisp
;;; holds the class definitions for data segmentation
;;; as well as the equation interpreting code

(in-package :user)
(shadowing-import '(screamer::defun screamer::either screamer::local))
(use-package '(:lisp :screamer :parmenides))

(defun clean ()
 (mapc #'remove-frame *class-list*)
 (mapc #'remove-frame *frame-list*)
 )

(defun debug ()
  (mapc #'pp-frame *frame-list*))

;;; Allows the calls
;;; (my-send object message arguments) to send a message to an object
;;; (my-send object slot facet) to get the value of the facet of an object
;;; (my-send object slot) to get the value of the slot of an object
;;; (my-send object slot facet value) to set the value of the facet of an object
;;; (my-send object slot value) to set the value of the slot of an object
(defun my-send (frame slot &rest rest)
  (cond
   ((methodp frame slot)
    (let ((self frame)
          (message slot))
      (declare (special self message))
      (apply (get-facet frame slot 'core) rest)))
   ((null rest) ; GET-SLOT OR GET-VALUE
    (if (has-facets frame slot)
        (get-value frame slot)
        (get-slot frame slot)))
   ((eq 1 (length rest)) ;GET-FACET OR SET-SLOT OR SET-VALUE
    (if (has-facets frame slot)
        (if (symbolp (car rest))
            (multiple-value-bind (value existence)
                                 (get-facet frame slot (car rest))
              (if existence
                  value
                  (set-value-demons frame slot (car rest))))
            (set-value-demons frame slot (car rest)))
        (set-slot frame slot (car rest))))
   (t ;SET-FACET
     (set-facet-demons frame slot (car rest) (cadr rest)))))

;;; Tests whether a slot is a method
(defun methodp (frame slot)
  (get-facet frame slot 'methodp))
```

```
;;; Defines a method attached to the slot of an object
(defmacro my-defmethod (frame slot name parameters &rest core)
  (let ((method `(defun ,(if name name (gensym "method"))
                        ,parameters
                   ,@core)))
    (if (get-slot frame slot)
        (progn
         (set-facet frame slot 'methodp t)
         `(set-facet ,frame ',slot 'core ,method))
        (add-slot frame slot
                  `(methodp t core ,method)))))

;;; Returns the function code implementing the method to answer a message
(defun get-method (frame method)
  (symbol-function (get-facet frame method 'core)))

;;;;;;;;;;;;;;;;;;

; CLASSES

;;;;;;;;;;;;;;;;;;

(def-frame has-shapes (:is-a relation)
 :has-inverses t
 :inverse-name from-signal)

(def-frame differential-of (:is-a relation)
 :has-inverses t
 :inverse-name integral-of)

(defun update-signal ()
  (update-shape-list)
  (update-zero-value))

;;; calculates the initial value of a segment
(defun update-zero-value ()
  (let* ((constant (get-value frame :has-constant))
         (c (if constant (get-value constant :constant) 0))
         (exponentials (get-value frame :has-exponentials))
         (ms (mapcar #'(lambda (exp) (get-value exp :max)) exponentials))
         (sinusoids (get-value frame :has-sinusoids))
         (as (mapcar #'(lambda (sim) (*v (get-value sin :amplitude)
                                         (sin (get-value sin :phase))))
                     sinusoids))
         (all (cons c (append ms as)))
         )
    (set-value frame :zero-value `(+v ,@all))))

;;; Posts an initial condition constraint
(defun check-zero-value (signal zero)
  (format t "~%(assert! (=v ~A ~A))" zero (get-value signal :zero-value)))
```

```
;;; Posts the new shape in its proper list within the segment
(defun update-shape-list ()
  (let ((newshape (first newval)))
    (cond
     ((isa-instance newshape 'constant)
      (set-value frame :has-constant newshape))
     ((isa-instance newshape 'linear)
      (set-value frame :has-linear newshape))
     ((isa-instance newshape 'exponential)
      (set-value frame :has-exponentials (list newshape)))
     ((isa-instance newshape 'sinusoid)
      (set-value frame :has-sinusoids (list newshape)))
     )))

;;; SEGMENT
(def-frame segment ()
  :name (:value 'unknown)
  :has-shapes (:value () :post-if-set '(update-segment))
  :has-constant (:value ())
  :has-linear (:value ())
  :has-exponentials (:value ())
  :has-sinusoids (:value ())
  :differential-of (:value ())
  :integral-of (:value ())
  :has-differential (:value 'unknown)
  :zero-value (:value 'unknown)
  )

(my-defmethod
 segment derive () ()
 (let ((linear (get-value self :has-linear))
       (exponentials (get-value self :has-exponentials))
       (sinusoids (get-value self :has-sinusoids))
       (derivation (make-segment
                    (gensym "segment"))))
   (add-to-facet-demons derivation :has-shapes :value (my-send linear 'derive))
   (mapcar #'(lambda (exponential)
                             (add-to-facet-demons derivation :has-shapes :value (my-send expo
'derive)))
           exponentials)
   (mapcar #'(lambda (sinusoid)
               (add-to-facet-demons derivation :has-shapes :value (my-send sinusoid 'derive)))
           sinusoids)
   (set-value self :has-differential derivation)
   ))

(defun check-derivation (segment)
  (let ((derivation (first (get-value segment :integral-of))))
    (cond
     (derivation
      (my-send segment 'derive)
      (compare-segments derivation (get-value segment :has-differential)))
     (t ())
     )))
```

```
;;; SHAPE
(def-frame shape (cache :*ALL*)
  :name (:value 'unknown)
  :from-segment (:value ()))

(my-defmethod
 shape copy () ()
 (let ((copy (copy-frame self)))
   (set-value copy :from-segment ())
   ))

;;; CONSTANT
(setq constant
      (def-frame constant (:is-a shape)
        :constant (:value 'unknown)))

(my-defmethod
 constant derive () ()
 ())

(my-defmethod
 constant minus () ()
 (make-constant
  (gensym "constant")
  :constant `(:value (*v -1.0 ,(get-value self :constant)))
  ))

(my-defmethod
 constant multiply () (variable)
 (make-constant
  (gensym "constant")
  :constant `(:value (*v ,(get-value self :constant) ,variable))
  ))

(my-defmethod
 constant divide () (variable)
 (make-constant
  (gensym "constant")
  :constant `(:value (/v ,(get-value self :constant) ,variable))
  ))

;;; LINEAR
(setq linear
      (def-frame linear (:is-a shape)
        :slope (:value 'unknown)))

(my-defmethod
 linear derive () ()
 (make-constant
  (gensym "constant")
  :constant `(:value ,(get-value self :slope))
  ))
```

```
(my-defmethod
 linear minus () ()
 (make-linear
  (gensym "linear")
  :slope `(:value (*v -1.0 ,(get-value self :slope)))
  ))

(my-defmethod
 linear multiply () (variable)
 (make-linear
  (gensym "linear")
  :slope `(:value (*v ,(get-value self :slope) ,variable))
  ))

(my-defmethod
 linear divide () (variable)
 (make-linear
  (gensym "linear")
  :slope `(:value (/v ,(get-value self :slope) ,variable))
  ))

;;; EXPONENTIAL
(setq exponential
    (def-frame exponential (:is-a shape)
      :max (:value 'unknown)
      :time-constant (:value 'unknown :fit 'unknown :precision 'unknown)))

(my-defmethod
 exponential derive () ()
 (make-exponential
  (gensym "exponential")
  :max `(:value (*v -1.0 (/v  ,(get-value self :max) ,(get-value self :time-constant))))
  :time-constant `(:value ,(get-value self :time-constant)
                   :fit ,(get-facet self :time-constant :fit)
                   :precision ,(get-facet self :time-constant :precision))
  ))

(my-defmethod
 exponential minus () ()
 (make-exponential
  (gensym "exponential")
  :max `(:value (*v -1.0 ,(get-value self :max)))
  :time-constant `(:value ,(get-value self :time-constant)
                   :fit ,(get-facet self :time-constant :fit)
                   :precision ,(get-facet self :time-constant :precision))
  ))
```

```
(my-defmethod
 exponential multiply () (variable)
 (make-exponential
  (gensym "exponential")
  :max `(:value (*v ,(get-value self :max) ,variable))
  :time-constant `(:value ,(get-value self :time-constant)
                   :fit ,(get-facet self :time-constant :fit)
                   :precision ,(get-facet self :time-constant :precision))
  ))

(my-defmethod
 exponential divide () (variable)
 (make-exponential
  (gensym "exponential")
  :max `(:value (/v ,(get-value self :max) ,variable))
  :time-constant `(:value ,(get-value self :time-constant)
                   :fit ,(get-facet self :time-constant :fit)
                   :precision ,(get-facet self :time-constant :precision))
  ))

;;; SINUSOID
(setq sinusoid
    (def-frame sinusoid (:is-a shape)
      :amplitude (:value 'unknown)
      :frequency (:value 'unknown :fit 'unknown :precision 'unknown)
      :phase (:value 'unknown :fit 'unknown :precision 'unknown)))

(setq pi/2 (/ pi 2))

(my-defmethod
 sinusoid derive () ()
 (make-sinusoid
  (gensym "sinusoid")
  :amplitude `(:value (*v  ,(get-value self :amplitude) ,(get-value self :frequency)))
  :frequency `(:value ,(get-value self :frequency)
                :fit ,(get-facet self :frequency :fit)
                :precision ,(get-facet self :frequency :precision))
  :phase `(:value (-v pi/2 ,(get-value self :phase))
                :fit (-v pi/2 ,(get-facet self :frequency :fit))
                :precision (-v pi/2 ,(get-facet self :frequency :precision)))
  ))

(my-defmethod
 sinusoid minus () ()
 (make-sinusoid
  (gensym "sinusoid")
  :amplitude `(:value (*v -1.0 ,(get-value self :amplitude)))
  :frequency `(:value ,(get-value self :frequency)
                :fit ,(get-facet self :frequency :fit)
                :precision,(get-facet self :frequency :precision))
  :phase `(:value ,(get-value self :phase)
             :fit ,(get-facet self :frequency :fit)
             :precision ,(get-facet self :frequency :precision))
  ))
```

```
(my-defmethod
 sinusoid multiply () (variable)
 (make-sinusoid
  (gensym "sinusoid")
  :amplitude `(:value (*v ,(get-value self :amplitude) ,variable))
  :frequency `(:value ,(get-value self :frequency)
                      :fit ,(get-facet self :frequency :fit)
                      :precision,(get-facet self :frequency :precision))
  :phase `(:value ,(get-value self :phase)
                  :fit ,(get-facet self :frequency :fit)
                  :precision ,(get-facet self :frequency :precision))
  ))

(my-defmethod
 sinusoid divide () (variable)
 (make-sinusoid
  (gensym "sinusoid")
  :amplitude `(:value (/v ,(get-value self :amplitude) ,variable))
  :frequency `(:value ,(get-value self :frequency)
                      :fit ,(get-facet self :frequency :fit)
                      :precision,(get-facet self :frequency :precision))
  :phase `(:value ,(get-value self :phase)
                  :fit ,(get-facet self :frequency :fit)
                  :precision ,(get-facet self :frequency :precision))
  ))

;;; EQUATION
(def-frame equation ()
  :LHS (:value ())
  :RHS (:value ()))

;;; EXPRESSION
(def-frame expression ()
  :operation (:value 'unknown)
  :arg1 (:value 'unknown)
  :arg2 (:value 'unknown))
```

```
;;;;;;;;;;;;;;;;;

; INSTANCES FOR THE SIMPLE EXAMPLE WITH SEGMENT #1

;;;;;;;;;;;;;;;;;


;;;;OMEGA

(assert! (realv (setq a (make-variable 'a))))

(setq omega-linear
    (make-linear
     'omega-linear
     :name '(:value omega-linear)
     :slope '(:value a)
    ))

(setq omega
    (make-segment
     'omega
     :name '(:value omega)
     ))

(add-to-facet-demons 'omega :has-shapes :value 'omega-linear)

;;;;Y-DOT-R

(assert! (realv (setq tr (make-variable 'tr))))
(assert! (realv (setq br (make-variable 'br))))

(setq y-dot-r-exponential
    (make-exponential
     'y-dot-r-exponential
     :name '(:value y-dot-r-exponential)
     :max '(:value (/v br tr))
     :time-constant '(:value tr)))

(assert! (realv (setq cr (make-variable 'cr))))

(setq y-dot-r-constant
    (make-constant
     'y-dot-r-constant
     :name '(:value y-dot-r-constant)
     :constant '(:value cr)))

(setq y-dot-r
    (make-segment
     'y-dot-r
     :name '(:value y-dot-r)
     ))

(add-to-facet-demons 'y-dot-r :has-shapes :value 'y-dot-r-exponential)
(add-to-facet-demons 'y-dot-r :has-shapes :value 'y-dot-r-constant)
```

```
;;;;Y-R

(setq y-r-exponential
    (make-exponential
     'y-r-exponential
     :name '(:value y-r-exponential)
     :max '(:value (*v -1.0 br))
     :time-constant '(:value tr)))

(setq y-r-constant
    (make-constant
     'y-r-constant
     :name '(:value y-r-constant)
     :constant '(:value br)))

(setq y-r-linear
    (make-linear
     'y-r-linear
     :name '(:value y-r-linear)
     :slope '(:value cr)
    ))

(setq y-r
    (make-segment
     'y-r
     :name '(:value y-r)
     ))
;;;     :zero-value '(:value 0)))

(add-to-facet-demons 'y-r :has-shapes :value 'y-r-exponential)
(add-to-facet-demons 'y-r :has-shapes :value 'y-r-constant)
(add-to-facet-demons 'y-r :has-shapes :value 'y-r-linear)
(set-value-demons 'y-r :integral-of 'y-dot-r)

;;;;;;;
; EQ1
;;;;;;;

(assert! (realv (setq tr (make-variable 'tr))))

(setq eq1
    (make-equation
     'eq1
     :LHS '(:value y-dot-r)
     :RHS `(:value ,(make-expression
                (gensym "expression")
                :operation '(:value -)
                :arg1 '(:value omega)
                :arg2 `(:value ,(make-expression
                            (gensym "expression")
                            :operation '(:value /)
                            :arg1 '(:value y-r)
                            :arg2 '(:value tr)))))))
```

```
;;;;;;;;;;;;;;;;;;

; LISTS

;;;;;;;;;;;;;;;;;;

(setq *class-list*
    '(
      has-shapes
      from-segment
      differential-of
      integral-of
      segment
      shape
      constant
      linear
      exponential
      sinusoid
      expression
      equation
      ))

(setq *frame-list*
    '(
      omega-linear
      omega
      y-dot-r-exponential
      y-dot-r-constant
      y-dot-r
      y-r-exponential
      y-r-constant
      y-r-linear
      y-r
      eq1
      ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;

; EQUATION INTERPRETER

;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;
; ADD
;;;;;;;;

;;; Adds or substracts two shapes

(defun add-shapes (segment-out operation segment1 segment2)
 (let ((constant
       (add-constants operation
                  (get-value segment1 :has-constant)
                  (get-value segment2 :has-constant)))
      (linear
       (add-linears operation
                 (get-value segment1 :has-linear)
                 (get-value segment2 :has-linear)))
      (exponentials
       (add-exponentials operation
                    ()
                    (get-value segment1 :has-exponentials)
                    (get-value segment2 :has-exponentials)))
      (sinusoids
       (add-sinusoids operation
                  ()
                  (get-value segment1 :has-sinusoids)
                  (get-value segment2 :has-sinusoids))))
  (add-to-facet-demons segment-out :has-shapes :value constant)
  (add-to-facet-demons segment-out :has-shapes :value linear)
  (mapcar #'(lambda (exponential)
          (add-to-facet-demons segment-out :has-shapes :value exponential))
        exponentials)
  (mapcar #'(lambda (sinusoid)
          (add-to-facet-demons segment-out :has-shapes :value sinusoid))
        sinusoids)
 )
 segment-out)
```

```
;;; Adds or substracts two constants
(defun add-constants (operation shape1 shape2)
 (cond
  ((and shape1 shape2)
   (make-constant
    (gensym "constant")
    :slope `(:value
          ,(list
            operation
              (get-value shape1 :constant)
              (get-value shape2 :constant))
           )))
  (shape1 (my-send shape1 'copy))
  (shape2 (if (equal operation '-v)
        (my-send shape2 'minus)
        (my-send shape2 'copy)))
  ))

;;; Adds or substracts two linears
(defun add-linears (operation shape1 shape2)
 (cond
  ((and shape1 shape2)
   (make-linear
    (gensym "linear")
    :slope `(:value
          ,(list
            operation
              (get-value shape1 :slope)
              (get-value shape2 :slope))
           )))
  (shape1 (my-send shape1 'copy))
  (shape2 (if (equal operation '-v)
        (my-send shape2 'minus)
        (my-send shape2 'copy)))
  ))

;;; Returns the shapes multiplied by -1
(defun minus-all (shapes)
 (mapcar #'(lambda (shape) (my-send shape 'minus)) shapes))

(defun copy-all (shapes)
 (mapcar #'(lambda (shape) (my-send shape 'copy)) shapes))

(defun add-exponentials (operation list shapes1 shapes2)
 (cond
  ((null shapes1) (append list (minus-all shapes2)))
  (t (let* ((shape1 (car shapes1))
        (shape2 (compatible-exponential shape1 shapes2)))
     (add-exponentials operation
                (append (merge-1-to-1-exponentials shape1 shape2) list)
                (cdr shapes1)
                (remove shape2 shapes2))))
  ))
```

```
;;; Returns the first exponential in shapes2 compatible with shape1
(defun compatible-exponential (shape1 shapes2)
 (cond
  ((null shapes2) ())
  (t
   (cond
    ((equal (get-value shape1 :time-constant)
            (get-value (car shapes2) :time-constant))
     (car shapes2))
    (t
     (compatible-exponential shape1 (cdr shapes2)))
    ))
  ))

;;; Returns the compound exponential from expo1 and expo2
(defun merge-1-to-1-exponentials (expo1 expo2)
 (cond
  (expo2
   (list
    (make-exponential
     (gensym "exponential")
     :max `(:value
            ,(list
              operation
              (get-value expo1 :max)
              (get-value expo2 :max)))
     :time-constant '(:value (get-value expo1 :time-constant)))))
  (t (my-send expo1 'copy))
  ))

;;; Adds or substracts two compatible sinusoids
(defun add-sinusoids (operation list shapes1 shapes2)
 (cond
  ((null shapes1) (append list (minus-all shapes2)))
  (t (let* ((shape1 (car shapes1))
            (shape2 (compatible-sinusoid shape1 shapes2)))
       (add-sinusoids operation
                      (append (merge-1-to-1-sinusoids shape1 shape2) list)
                      (cdr shapes1)
                      (remove shape2 shapes2))))
  ))
```

```
;;; Returns the first sinusoid in shapes2 compatible with shape1
(defun compatible-sinusoid (shape1 shapes2)
 (cond
  ((null shapes2) ())
  (t
   (cond
    ((and
      (equal (get-value shape1 :frequency)
           (get-value (car shapes2) :frequency))
      (equal (get-value shape1 :phase)
           (get-value (car shapes2) :phase)))
     (car shapes2))
    (t
     (compatible-sinusoid shape1 (cdr shapes2)))
    ))
  ))

;;; Returns the compound sinusoid from sin1 and sin2
(defun merge-1-to-1-sinusoids (sin1 sin2)
 (cond
  (sin2
   (list
    (make-sinusoid
     (gensym "sinusoid")
     :amplitude `(:value
                  ,(list
                    operation
                    (get-value sin1 :amplitude)
                    (get-value sin2 :amplitude)))
     :frequency '(:value (get-value sin1 :frequency))
     :phase '(:value (get-value sin1 :phase)))))
  (t (my-send sin1 'copy))
  ))
```

```
;;;;;;;;;;;;
; REDUCE
;;;;;;;;;;;;

;;; Reduces expression to a single segment
(defun reduce-to-segment (expression)
 (cond
  ((isa-instance expression 'segment) expression)        ;;nothing to do for segments
  ((isa-instance expression 'expression)                 ;;for expressions
   (let ((operation (get-value expression :operation))
       (arg1 (get-value expression :arg1))
       (arg2 (get-value expression :arg2))
       (segment (make-segment (gensym "segment"))))
    (cond
     ((member operation '(+ -))                          ;;if + or -, arg1 and arg2 are segments
      (let ((op (if (equal operation '+)                 ;;or expressions
                '+v
                '-v)))
       (add-shapes segment
                op
                (reduce-to-segment arg1)
                (reduce-to-segment arg2))))
     ((member operation '(* /))                          ;;if * or /, arg1 is a segment
      (let ((shapes (get-value arg1 :has-shapes))        ;;and arg2 is a variable
          (op (if (equal operation '*)
                'mutiply
                'divide)))
       (mapcar #'(lambda (shape)
                (let ((newshape (my-send shape op arg2)))
                  (add-to-facet-demons segment :has-shapes :value newshape)))
              shapes))
      segment)
     )))
  (t (format t "~%Unknown expression type: ~A" expression))))

;;; Transforms a model equation into constraints
(defun interpret-equation (equation)
 (let
  ((lhs (reduce-to-segment (get-value equation :lhs)))
   (rhs (reduce-to-segment (get-value equation :rhs)))
   )
  (compare-segments lhs rhs)
  ))
```

```
;;;;;;;;;;;;
; COMPARE
;;;;;;;;;;;;

;;; Compares two segments shape by shape
(defun compare-segments (segment1 segment2)
 (let ((constant1 (get-value segment1 :has-constant))
       (constant2 (get-value segment2 :has-constant))
       (linear1 (get-value segment1 :has-linear))
       (linear2 (get-value segment2 :has-linear))
       (exponentials1 (get-value segment1 :has-exponentials))
       (exponentials2 (get-value segment2 :has-exponentials))
       (sinusoids1 (get-value segment1 :has-sinusoids))
       (sinusoids2 (get-value segment2 :has-sinusoids)))
   (compare-constants constant1 constant2)
   (compare-linears linear1 linear2)
   (compare-exponentials exponentials1 exponentials2)
   (compare-sinusoids sinusoids1 sinusoids2)
   ))

;;; Compares two constants, and generates appropriate constraints
(defun compare-constants (constant1 constant2)
 (cond
  ((not (or constant1 constant2)) ())
  (t (let ((c1 (if (isa-instance constant1 'constant)
                   (get-value constant1 :constant)
                   0))
           (c2 (if (isa-instance constant2 'constant)
                   (get-value constant2 :constant)
                   0)))
       (format t "~% (assert! (=v ~A ~A))" c1 c2)
;;;      (assert! (=v c1 c2))
       ))
  ))

;;; Compares two linears, and generates appropriate constraints
(defun compare-linears (linear1 linear2)
 (cond
  ((not (or linear1 linear2)) ())
  (t (let ((s1 (if (isa-instance linear1 'linear)
                   (get-value linear1 :slope)
                   0))
           (s2 (if (isa-instance linear2 'linear)
                   (get-value linear2 :slope)
                   0)))
       (format t "~% (assert! (=v ~A ~A))" s1 s2)
;;;      (assert! (=v s1 s2))
       ))
  ))

;;; Compares two exponentials, and generates appropriate constraints
(defun compare-exponentials (shapes1 shapes2)
 (cond
  ((null shapes1)
```

```
   (mapcar #'(lambda (expo) (assert-expo-constraint expo ()))
           shapes2))
   (t (let* ((shape1 (car shapes1))
             (shape2 (smart-compatible-exponential shape1 shapes2)))
        (assert-expo-constraint shape1 shape2)
        (compare-exponentials
         (cdr shapes1)
         (remove shape2 shapes2))))
   ))

;;; Checks if a variable is a boolena
(defun booleanv (x)
  (memberv x '(t nil)))

;;; Checks if two vriables are identical
(defun same-variablep (x y)
  (if (known? (equalv x y)) t ()))

;;; Returns the first compatible exponential in shapes 2 (or numerically
;;; almost compatible) with shape1
(defun smart-compatible-exponential (shape1 shapes2)
  (let ((shape2 (car shapes2)))
    (cond
     ((null shapes2) ())
     (t
      (cond
       ((almost-equal shape1 shape2 :time-constant) shape2)
       (t (smart-compatible-exponential shape1 (cdr shapes2)))
       ))
     )))

;;; Checks if two curve fit variables are numerically almost compatible
(defun almost-equal (shape1 shape2 slot)
  (let ((var1 (get-value shape1 slot))
        (fit1 (get-facet shape1 slot :fit))
        (prec1 (get-facet shape1 slot :precision))
        (var2 (get-value shape2 slot))
        (fit2 (get-facet shape2 slot :fit))
        (prec2 (get-facet shape2 slot :precision)))
    (cond
     ((equal var1 var2) t)
     ((or (>= (+ fit1 prec1) (- fit2 prec2))
          (>= (+ fit2 prec2) (- fit1 prec1)))
      (format t "~% (assert! (=v ~A ~A))" var1 var2)
      ;;;    (assert! (=v var1 var2))
      t)
     (t ()))))
```

```
;;; Assert the constraints that enforce the two exponentials to be identical
(defun assert-expo-constraint (expo1 expo2)
 (let ((m1 (get-value expo1 :max))
       (m2 (if expo2
               (get-value expo2 :max)
               0)))
   (format t "~% (assert! (=v ~A ~A))" m1 m2)
;;;   (assert! (=v m1 m2))
   ))

;;; Compares two sinusoids, and generates appropriate constraints
(defun compare-sinusoids (shapes1 shapes2)
 (cond
  ((null shapes1)
   (mapcar #'(lambda (expo) (assert-sin-constraint expo ()))
           shapes2))
  (t (let* ((shape1 (car shapes1))
            (shape2 (smart-compatible-sinusoid shape1 shapes2)))
       (assert-sin-constraint shape1 shape2)
       (compare-sinusoids
        (cdr shapes1)
        (remove shape2 shapes2))))
  ))

;;; Returns the first compatible sinusoid in shapes 2 (or numerically
;;; almost compatible) with shape1
(defun smart-compatible-sinusoid (shape1 shapes2)
 (let ((shape2 (car shapes2)))
  (cond
   ((null shapes2) ())
   (t
    (cond
     ((and
       (almost-equal shape1 shape2 :frequency)
       (almost-equal shape1 shape2 :phase))
      (car shapes2))
     (t
      (smart-compatible-sinusoid shape1 (cdr shapes2)))
     ))
   )))

;;; Assert the constraints that enforce the two sinusoids to be identical
(defun assert-sin-constraint (sin1 sin2)
 (let ((a1 (get-value sin1 :amplitude))
       (a2 (if sin2
               (get-value sin2 :amplitude)
               0)))
   (format t "~% (assert! (=v ~A ~A))" a1 a2)
;;;   (assert! (=v a1 a2))
   ))
```

```
;;;;;;;;;;
;;;;;;;;;;
; TEST
;;;;;;;;;;
;;;;;;;;;;

(defun eq-test () (interpret-equation eq1))

(defun deriv-test () (check-derivation y-r0))

(assert! (realv (setq d (make-variable 'd))))
(assert! (realv (setq e (make-variable 'e))))
(assert! (realv (setq t2 (make-variable 't2))))

;;; Y-DOT-R0
(setq y-dot-r0-exponential
    (make-exponential
     'y-dot-r0-exponential
     :name '(:value y-dot-r0-exponential)
     :max '(:value e)
     :time-constant '(:value t2 :fit 7 :precision .1)))

(setq y-dot-r0-constant
    (make-constant
     'y-dot-r0-constant
     :name '(:value y-dot-r0-constant)
     :constant '(:value d)))

(setq y-dot-r0
    (make-segment
     'y-dot-r0
     :name '(:value y-dot-r0)
     ))

(add-to-facet-demons 'y-dot-r0 :has-shapes :value 'y-dot-r0-exponential)
(add-to-facet-demons 'y-dot-r0 :has-shapes :value 'y-dot-r0-constant)

(assert! (realv (setq t1 (make-variable 't1))))
(assert! (realv (setq a (make-variable 'a))))
(assert! (realv (setq b (make-variable 'b))))
(assert! (realv (setq c (make-variable 'c))))

;;; Y-R0
(setq y-r0-exponential
    (make-exponential
     'y-r0-exponential
     :name '(:value y-r0-exponential)
     :max '(:value c)
     :time-constant '(:value t1 :fit 7 :precision .1)))

(setq y-r0-constant
    (make-constant
     'y-r0-constant
     :name '(:value y-r0-constant)
     :constant '(:value a)))
```

```
(setq y-r0-linear
    (make-linear
     'y-r0-linear
     :name '(:value y-r0-linear)
     :slope '(:value b)
    ))

(setq y-r0
    (make-segment
     'y-r0
     :name '(:value y-r0)
     ))

(add-to-facet-demons 'y-r0 :has-shapes :value 'y-r0-exponential)
(add-to-facet-demons 'y-r0 :has-shapes :value 'y-r0-constant)
(add-to-facet-demons 'y-r0 :has-shapes :value 'y-r0-linear)
(set-value-demons 'y-r0 :integral-of 'y-dot-r0)
```

APPENDIX D

Chair Example Constraint Propagation Code

This code shows the Screamer, Parmenides and Lisp code used in the constraint propagation phase of the normal continuous chair example.

```
(in-package :user)
;;; added two problem definitions
(shadowing-import '(screamer::defun screamer::either screamer::local))
(use-package '(:lisp :screamer))
;;; removed old shadow
;;; (shadowing-import '(screamer::defun))

;;; Almost setting the value of a variable
(defun === (var val)
  (assert! (<=v var (* val (+ 1 (* *fuzz* 10.0)))))
  (assert! (>=v var (* val (- 1 (* *fuzz* 10.0))))))

;;; Display all data
(defun show-all ()
 (show-fits1)
 (show-fits2)
 (show-model))

;;; Display data for first experiment
(defun show-fits1 ()
 (let ((old-fuzz *fuzz*)
       (*fuzz* 0.01))
   (ed-beep)
   (ed-beep)
   (format t "~% a1 = ~4,2F" a1)
   (format t "~% b1 = ~4,2F" b1)
   (format t "~% c1 = ~4,2F" c1)
   (format t "~% d1 = ~4,2F" d1)
   (format t "~% e1 = ~4,2F" e1)
   (format t "~% f1 = ~4,2F" f1)
   (format t "~% g1 = ~4,2F" g1)
   (format t "~% h1 = ~4,2F" h1)
   (format t "~% i1 = ~4,2F" i1)
   (format t "~% j1 = ~4,2F" j1)
   (format t "~% kk1 = ~4,2F" kk1)
   (format t "~% l1 = ~4,2F" l1)
   (format t "~% m1 = ~4,2F" m1)
   (format t "~% n1 = ~4,2F" n1)
   (format t "~% o1 = ~4,2F" o1)
   (format t "~% p1 = ~4,2F" p1)
   (format t "~% q1 = ~4,2F" q1)
   (format t "~% r1 = ~4,2F" r1)
   (format t "~% s1 = ~4,2F" s1)
   (format t "~% t11 = ~4,2F" t11)
   (format t "~% t21 = ~4,2F" t21)
   (format t "~% tt1 = ~4,2F" tt1)
   ))
```

```
;;; Display data for second experiment
(defun show-fits2 ()
  (let ((old-fuzz *fuzz*)
        (*fuzz* 0.01))
    (ed-beep)
    (ed-beep)
    (format t "~% a2 = ~4,2F" a2)
    (format t "~% b2 = ~4,2F" b2)
    (format t "~% c2 = ~4,2F" c2)
    (format t "~% d2 = ~4,2F" d2)
    (format t "~% e2 = ~4,2F" e2)
    (format t "~% f2 = ~4,2F" f2)
    (format t "~% g2 = ~4,2F" g2)
    (format t "~% h2 = ~4,2F" h2)
    (format t "~% i2 = ~4,2F" i2)
    (format t "~% j2 = ~4,2F" j2)
    (format t "~% kk2 = ~4,2F" kk2)
    (format t "~% l2 = ~4,2F" l2)
    (format t "~% m2 = ~4,2F" m2)
    (format t "~% n2 = ~4,2F" n2)
    (format t "~% o2 = ~4,2F" o2)
    (format t "~% p2 = ~4,2F" p2)
    (format t "~% q2 = ~4,2F" q2)
    (format t "~% r2 = ~4,2F" r2)
    (format t "~% s2 = ~4,2F" s2)
    (format t "~% t12 = ~4,2F" t12)
    (format t "~% t22 = ~4,2F" t22)
    (format t "~% tt2 = ~4,2F" tt2)
    ))

;;; Display model parameters
(defun show-model ()
  (let ((old-fuzz *fuzz*)
        (*fuzz* 0.01))
    (ed-beep)
    (ed-beep)
    (format t "~% tr = ~4,2F" tr)
    (format t "~% tl = ~4,2F" tl)
    (format t "~% tcap = ~4,2F" tcap)
    (format t "~% w1 = ~4,2F" w1)
    (format t "~% w2 = ~4,2F" w2)
    (format t "~% k = ~4,2F" k)
    ))
```

```
;;; Define curve fit parameters for first experiment
(assert! (realv (setq a1 (make-variable 'a1))))
(assert! (realv (setq b1 (make-variable 'b1))))
(assert! (realv (setq c1 (make-variable 'c1))))
(assert! (realv (setq d1 (make-variable 'd1))))
(assert! (realv (setq e1 (make-variable 'e1))))
(assert! (realv (setq f1 (make-variable 'f1))))
(assert! (realv (setq g1 (make-variable 'g1))))
(assert! (realv (setq h1 (make-variable 'h1))))
(assert! (realv (setq i1 (make-variable 'i1))))
(assert! (realv (setq j1 (make-variable 'j1))))
(assert! (realv (setq kk1 (make-variable 'kk1))))
(assert! (realv (setq l1 (make-variable 'l1))))
(assert! (realv (setq m1 (make-variable 'm1))))
(assert! (realv (setq n1 (make-variable 'n1))))
(assert! (realv (setq o1 (make-variable 'o1))))
(assert! (realv (setq p1 (make-variable 'p1))))
(assert! (realv (setq q1 (make-variable 'q1))))
(assert! (realv (setq r1 (make-variable 'r1))))
(assert! (realv (setq s1 (make-variable 's1))))
(assert! (realv (setq tt1 (make-variable 'tt1))))
(assert! (realv (setq t11 (make-variable 't11))))
(assert! (realv (setq t21 (make-variable 't21))))

;;; Define curve fit parameters for second experiment
(assert! (realv (setq a2 (make-variable 'a2))))
(assert! (realv (setq b2 (make-variable 'b2))))
(assert! (realv (setq c2 (make-variable 'c2))))
(assert! (realv (setq d2 (make-variable 'd2))))
(assert! (realv (setq e2 (make-variable 'e2))))
(assert! (realv (setq f2 (make-variable 'f2))))
(assert! (realv (setq g2 (make-variable 'g2))))
(assert! (realv (setq h2 (make-variable 'h2))))
(assert! (realv (setq i2 (make-variable 'i2))))
(assert! (realv (setq j2 (make-variable 'j2))))
(assert! (realv (setq kk2 (make-variable 'kk2))))
(assert! (realv (setq l2 (make-variable 'l2))))
(assert! (realv (setq m2 (make-variable 'm2))))
(assert! (realv (setq n2 (make-variable 'n2))))
(assert! (realv (setq o2 (make-variable 'o2))))
(assert! (realv (setq p2 (make-variable 'p2))))
(assert! (realv (setq q2 (make-variable 'q2))))
(assert! (realv (setq r2 (make-variable 'r2))))
(assert! (realv (setq s2 (make-variable 's2))))
(assert! (realv (setq tt2 (make-variable 'tt2))))
(assert! (realv (setq t12 (make-variable 't12))))
(assert! (realv (setq t22 (make-variable 't22))))
```

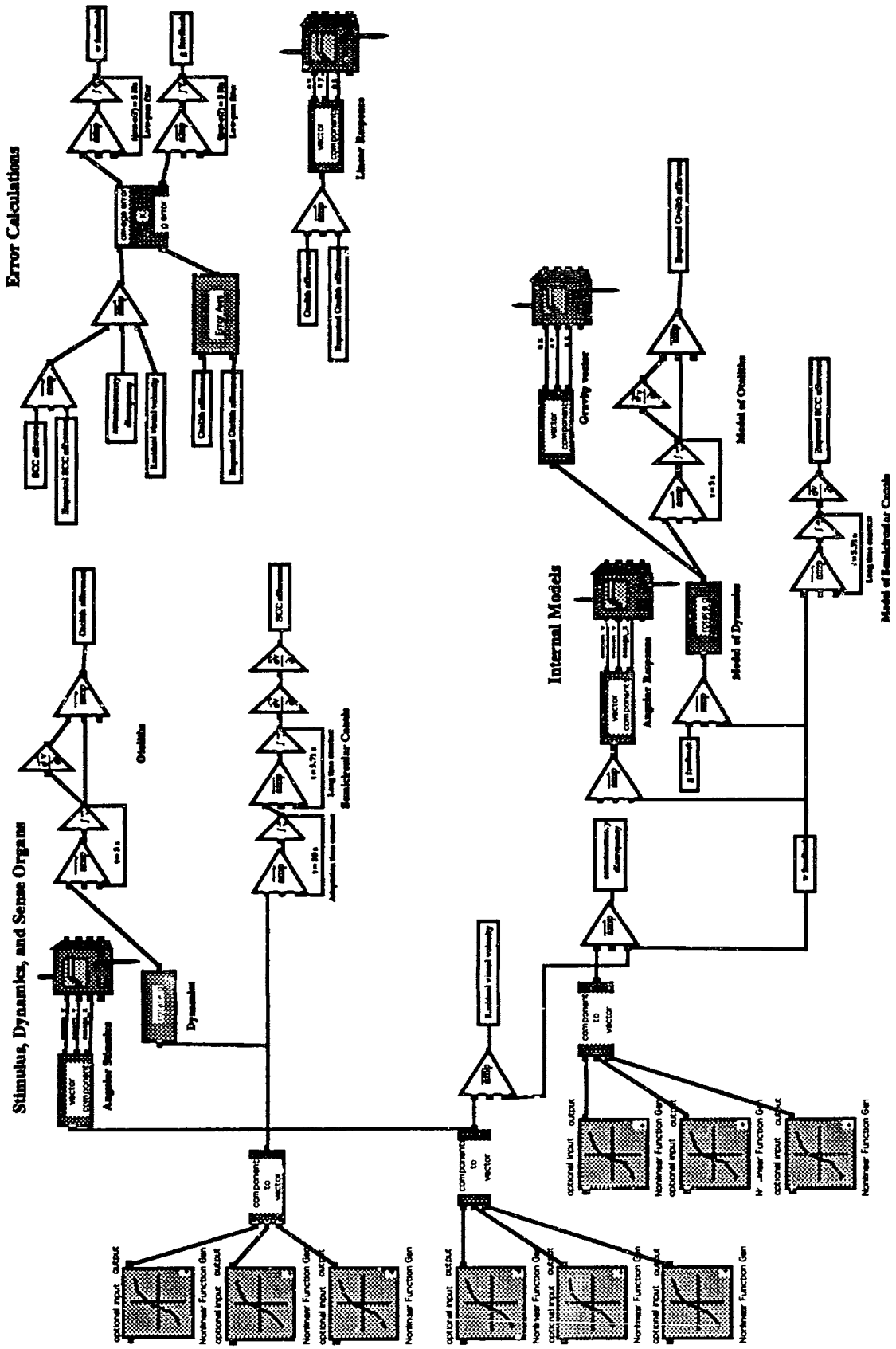
```
;;; Define model parameters
(assert! (realv (setq tr (make-variable 'tr))))
(assert! (realv (setq tl (make-variable 'tl))))
(assert! (realv (setq tcap (make-variable 'tcap))))
(assert! (realv (setq w1 (make-variable 'w1))))
(assert! (realv (setq w2 (make-variable 'w2))))
(assert! (realv (setq k (make-variable 'k))))

;;; Constraints for first experiment
(assert! (=v a1 (/v c1 tr)))
(assert! (=v f1 (*v -1.0 (/v d1 tr))))
(assert! (=v e1 (*v -1.0 (/v b1 tr))))
(assert! (=v a1 (/v h1 tl)))
(assert! (=v j1 (*v -1.0 (/v g1 tl))))
(assert! (=v i1 (+v (*v w1 e1) (*v w2 j1))))
(assert! (=v m1 (+v (*v w1 f1) (*v w2 kk1))))
(assert! (=v n1 (/v p1 tcap)))
(assert! (=v s1 (*v -1.0 (/v q1 tcap))))
(assert! (=v tt1 (+v (*v -1.0 (/v r1 tcap)) o1)))
(assert! (=v n1 (*v k l1)))
(assert! (=v o1 (*v -1.0 k tt1)))
(assert! (=v m1 s1))
(assert! (=v c1 e1))
(assert! (=v (*v -1.0 (/v d1 t11)) f1))
(assert! (=v h1 j1))
(assert! (=v (*v -1.0 (/v i1 t11)) kk1))
(assert! (=v (*v -1.0 (/v q1 t11)) s1))
(assert! (=v (*v -1.0 (/v r1 t21)) tt1))

;;; Output curve fit for first experiment
(assert! (=v a1 0.2))
(assert! (=v n1 (* 0.7 (/ 3 3.0))))
(assert! (=v o1 (*v -1.0 n1)))
(assert! (=v t21 7.0))

;;; Model parameter constraints
(assert! (>=v k 1))
(assert! (<=v k 3))

(assert! (>=v tr 6.0))
(assert! (<=v tr 8.0))

(assert! (>=v tl 6.0))
(assert! (<=v tl 8.0))

(assert! (>=v tcap 6.0))
(assert! (<=v tcap 8.0))

(assert! (>=v w1 (/ 1 6.0)))
(assert! (<=v w1 (/ 2 3.0)))

(assert! (>=v w2 (/ 1 3.0)))
(assert! (<=v w2 (/ 4 3.0)))
```

```
;;; Display progress
(show-fits1)
(show-model)

;;; Constraints for second experiment
(assert! (=v a2 (/v c2 tr)))
(assert! (=v f2 (*v -1.0 (/v d2 tr))))
(assert! (=v e2 (*v -1.0 (/v b2 tr))))
(assert! (=v a2 (/v h2 tl)))
(assert! (=v j2 (*v   .0 (/v g2 tl))))
(assert! (=v l2 (+v (*v w2 e2) (*v w1 j2))))
(assert! (=v m2 (+v (*v w2 f2) (*v w1 kk2))))
(assert! (=v n2 (/v p2 tcap)))
(assert! (=v s2 (*v -1.0 (/v q2 tcap))))
(assert! (=v tt2 (+v (*v -1.0 (/v r2 tcap)) o2)))
(assert! (=v n2 (*v k l2)))
(assert! (=v o2 (*v -1.0 k tt2)))
(assert! (=v n2 s2))
(assert! (=v c2 e2))
(assert! (=v (*v -1.0 (/v d2 t12)) f2))
(assert! (=v h2 j2))
(assert! (=v (*v -1.0 (/v i2 t12)) kk2))
(assert! (=v (*v -1.0 (/v q2 t12)) s2))
(assert! (=v (*v -1.0 (/v r2 t22)) tt2))

;;; Output curve fit for second experiment
(assert! (=v a2 0.2))
(assert! (=v n2 (* 2.8 (/ 1 3.0))))
(assert! (=v o2 (*v -1.0 n2)))
(assert! (=v t22 21.0))

;;; Display all results
(show-all)
```

APPENDIX E

EXTEND Modeling Code

This code implements the vestibular model and all the necessary sub-components in Extend 1.1j. The model is easy to read as a functional diagram, but not very efficient.


Error Calculations
Stimulus, Dynamics, and Sense Organs
Internal Models

Components to Vector

This block accepts up to three scalars as its input. It then scales these scalars by the scale factors entered via the dialog box. The output is a three element vector.

Written by Dan Merfeld October 1989
Modified by Dan Merfeld on 7/21/90

```
real dummy [];


** This message occurs for each step in the simulation.
on simulate
{

** put the three components into a vector
dummy[0] = xIn*gain1;
dummy[1] = yIn*gain2;
dummy[2] = zIn*gain3;

vectorOut = passArray(dummy);
}


** If the dialog data is inconsistent for simulation, abort.
on checkdata
{

}


** Initialize any simulation variables.
on initsim

{

makearray(dummy, 3);

}
on createBlock
{
gain1=1.0;
gain2=1.0;
gain3=1.0;
}

** User clicked the dialog HELP button.
on help
{
showHelp();
}
```

Error Axis

This block takes 2 three component vectors and determines the magnitude of the angle between the vectors and the direction of rotation required to align the vectors. This information is output as a three component vector.

Written by Dan Merfeld October 1989
Modified by: Dan Merfeld 7/21/90

```
** cross product */
** totalSteps, curStep, totalTime, curTime, deltaTime */
** are defined by the system */

real                    tempOut[],mag,mag1,mag2,dot,angle;
real                    temp1In[],temp2In[];


on checkdata
{

}

on initsim
{
makearray(tempOut,3);
makearray(temp1In,3);
makearray(temp2In,3);
}

** simulation part */
on simulate
        {
        GetPassedArray(Vector1In,temp1In);
        GetPassedArray(Vector2In,temp2In);

** calculate the cross product of the two vectors
        tempOut[0] = temp1In[1]*temp2In[2] - temp1In[2]*temp2In[1];
        tempOut[1] = temp1In[2]*temp2In[0] - temp1In[0]*temp2In[2];
        tempOut[2] = temp1In[0]*temp2In[1] - temp1In[1]*temp2In[0];

** calculate the magnitude of the cross product and the two input vectors
        mag=sqrt(tempOut[0]^2 + tempOut[1]^2 + tempOut[2]^2);
        mag1=sqrt(temp1In[0]^2 + temp1In[1]^2 + temp1In[2]^2);
        mag2=sqrt(temp2In[0]^2 + temp2In[1]^2 + temp2In[2]^2);

** calculate the dot product
        dot=temp1In[0]*temp2In[0]+temp1In[1]*temp2In[1]+temp1In[2]*temp2In[2];
        angle=180/3.1415927*acos(dot/(mag1*mag2));
        if(mag > 0)
        {
                tempOut[0]=tempOut[0]*angle/mag;
                tempOut[1]=tempOut[1]*angle/mag;
                tempOut[2]=tempOut[2]*angle/mag;
```

```
        }
        else
        {
                tempOut[0]=0;
                tempOut[1]=0;
                tempOut[2]=0;
        }
        vectorOut = PassArray(tempOut);
        }


on help
{
showHelp();
}
```

Non-Linear Function Generator

OVERVIEW:

Output = f(time or optional input) as linearly interpolated from points in user supplied data table.

- Input: default is simulation current time (i.e. "function generator mode") or is taken from input connector on block ("time invariant nonlinearity mode"). Specify mode with radio button.

- Function definition: enter as many input/output value pairs as required (up to 1500) to adequately define a continuous function. Values need not be in order, but will be sorted in order of the "Sort" button is pushed. Be sure the input values provided span the range of input values which will be encountered. Out of range input values will trigger an abort message during the simulation.

- Press "Sort" if data requires sorting.

- Optional Plot: push button to see an auto-scaled plot of the values in your data table.

DESCRIPTION:

Applications: This block will generate an arbitrary time function as an input to a simulation. Alternatively it can be used as a time invariant nonlinear system element. You can transfer computed function numerical values to the block data table using the clipboard. You need only specify the value of the function where the derivative changes. The block interpolates automatically. In many applications, the input is piecewise linear, so this block is easier to use than a "Plotter I/O" block, which requires data table values at equally spaced intervals in time.

Original Authors: Cheryl Blanford and Chuck Oman
MIT Man Vehicle Laboratory
Rm 37-219, Cambridge, MA 02139
with modifications by Alfy Riddle, Imagine That! 5/22/89

```
** Nonlinear Function Generator 2/20/89
** Original Authors: Cheryl Blanford and Chuck Oman
** MIT Man Vehicle Laboratory
** Rm 37-219, Cambridge, MA 02139

** modifications made to the original program:
**                      Alfy Riddle, Imagine That!, 4/10/89
**              eliminated Data Length parameter
**              installed 'validMax' procedure
**              installed interpolation in 'simulate' message which
**                      remembers last data point (for currentTime only!)
**
real dataonly[][2];
real xdata[],ydata[];
real dataMax, dataMin;          ** corrected for roundoff error
integer max;                    ** length of data array ANR 4/10/89
integer xPoint;                 ** for incrementing x index on CurrentTime input
integer x,y;
real tempx,tempy,xvalue,yvalue;
```

```
Procedure
validMax()    ** ANR 4/10/89
{
        ** count length of data array
        integer i;
        i=0;

        while( !noValue(data[i][0]) && i<1500 )
        {
                i++;
        }
        max = i;
}

on sort
{
        validMax();             ** ANR 4/10/89

** split data into 2 arrays for ease of sorting.
makearray(xdata,max);
makearray(ydata,max);

for(x = 0 ;x<max;x++)
 {xdata[x] = data[x][0];
  ydata[x] = data[x][1];}

for (x=0;x<max;x++)
  for (y=x+1;y<max;y++)
    if(xdata[y] < xdata[x])
    {
     tempx = xdata[y];
     tempy = ydata[y];
     xdata[y] = xdata[x];
     ydata[y] = ydata[x];
     xdata[x] = tempx;
     ydata[x] = tempy;
    }


**restore table, now properly sorted.
for (x = 0;x<max;x++ )
 {data[x][0] = xdata[x];
  data[x][1] = ydata[x];}
}

on choosetoplot
{
        integer k;
        validMax();             ** ANR 4/10/89

        makearray(xdata,max);                   ** ANR 4/10/89
        makearray(ydata,max);
```

```
        for (k=0;k<max;k++)
        {
                xdata[k] = data[k][0];
                ydata[k] = data[k][1];
        }

        ** install the axes
        ** modified 5/22/89, PlotNewScatter not required ANR
        installAxis(0, "Nonlinear Function Generator Characteristic",
                "INPUT", FALSE, xData[0],xData[max-1],
                        "OUTPUT", FALSE, 0.0, 1.0, "", 0, 0.0, 0.0,
                        blackpattern, blackcolor, max);
        installArray(0, 0, "Input", xdata, xData[0],xData[max-1],
                                        max, 0, blackPattern, cyanColor);
        installArray(0, 1, "Output", ydata, xData[0],xData[max-1],
                                        max, 0, blackPattern, cyanColor);

   makeScatter(0, 0);
   autoscaley(0);
   showPlot(0,"Nonlinear Function Generator Characteristic");
}

** This message occurs for each step in the simulation.
on simulate
{
        integer lastX,notDone;          ** ANR 4/10/89
        lastX = 0;              ** this is a starting point for the interpolation
                                        ** it keeps track of the last interpolated point

        **
        **      dataMax & dataMin are calculated on initSim ANR 5/16/89
        **  read in x value and interpolate to find the y value.
        **      'max' gives the number of data points & is set during initSim
        **
        if(timebutton==1)       ** ANR 5/22/89
        {
                xValue=currentTime;

                ** get current valid range of data (xPoint = -1 in initSim)
                while( (xValue >= data[xPoint+1][0]) && (xPoint+1 < max) )
                {
                        xPoint++;
                }

                if( xPoint+1 >= max )           ** data at last point or too high
                {
                        if( (xPoint == max-1) && (realAbs(xValue-data[xPoint][0]) <
1.0e-15) )
                                yOut = data[xPoint][1];
                        else                                    ** ERROR - above range
                        {
                                userError
                                ("Nonlinear function generator input too high to
interpolate");
```

```
                        abort;
                    }
                }
                else                                    ** data in or below range
                {
                    if( (xPoint >= 0) && (xPoint < max-1) )          ** interpolate
                    {
                        yValue = ((xValue - data[xPoint][0])*(data[xPoint+1][1] -
                            data[xPoint][1])/(data[xPoint+1][0]     -
data[xPoint][0]))
                            + data[xPoint][1];

                        yOut = yValue;
                    }
                    else                                ** data near first point or ERROR
                    {
                        if( (realAbs(xValue-data[0][0]) < 1.0e-15) )
                            yOut = data[0][1];
                        else                                ** ERROR - below range
                        {
                            userError
                            ("Nonlinear function generator input too low to
interpolate");
                            abort;
                        }
                    }
                }
            }
            else    ** for connector input (original code)
            {
                xvalue = xin;
                if(xvalue<dataMin)    ** avoids roundoff error msg first pt
                 {
                  usererror(
                  "nonlinear function generator input too low to interpolate");
                  abort;
                  }

                for x=0 to (max-1)
                {
                 if (xvalue > dataMax)          ** avoids roundoff error msg last pt
                        {
                        usererror
                        ("nonlinear function generator input too high to interpolate");
                        abort;
                        }

                 if (xvalue == data[x][0])
                   {
                    yout = data[x][1];
                    break;
                    }

                 if (xvalue < data[x][0])
```

```
                {
                yvalue = ((xvalue - data[x-1][0])*(data[x][1] -
                                data[x-1][1])/(data[x][0] - data[x-1][0]))
                                + data[x-1][1];
                yout = yvalue;
                break;
                }

               }
        }
}


on endsim
{
        pushPlotPic(0);
}



on checkdata
{

}


** Initialize any simulation variables.
on initsim
{
        validMax();             ** make sure we have this number ANR 4/10/89

        if( data[max-1][0] > 0 )                ** for roundoff error ANR 5/16/89
                dataMax = data[max-1][0]*1.000000001;
        else
                dataMax = data[max-1][0]*0.999999999;

        if( data[0][0] > 0 )
                dataMin = data[0][0]*0.999999999;
        else
                dataMin = data[0][0]*1.000000001;

        xPoint = -1;    ** for currentTime mode ANR 5/22/89
}


** User clicked the dialog HELP button.
on help
{
showHelp();


}
```

Rotate G

This block accepts angular velocity as its input. The initial position of gravity is input via a dialog box. A quaternion integrator is used to continually calculate the orientation of g with respect to the rotation body.

Written by Dan Merfeld October 1989
(This program took parts of a program written by Brad McGrath.)
Modified by Dan Merfeld 7/21/90

```
real dcos[],g_body[];
real e0,e1,e2,e3;
real w_xrad,w_yrad,w_zrad;
real omega[];
** This message occurs for each step in the simulation.
on simulate
{

**declare variables

real red,blue,pink,grey;
real k,z;

GetPassedArray(w_in,omega);

w_xrad = pi * (omega[0]/180);
w_yrad = pi * (omega[1]/180);
w_zrad = pi * (omega[2]/180);

** perform the quaternion calculations
red = -0.5*(e1*w_xrad + e2*w_yrad + e3*w_zrad);
blue= 0.5*(e0*w_xrad + e2*w_zrad - e3*w_yrad);
pink= 0.5*(e0*w_yrad + e3*w_xrad - e1*w_zrad);
grey= 0.5*(e0*w_zrad + e1*w_yrad - e2*w_xrad);

k = 0.9*(1/deltatime);
z = 1 - (e0^2 + e1^2 + e2^2 + e3^2);

e0 = e0 + deltatime*(red + k*z*e0);
e1 = e1 + deltatime*(blue + k*z*e1);
e2 = e2 + deltatime*(pink + k*z*e2);
e3 = e3 + deltatime*(grey + k*z*e3);

** Calculate dirn cosines from quarternions
dcos[0] =e0^2+e1^2-e2^2-e3^2;
dcos[1] =2*(e1*e2+e0*e3);
dcos[2] =2*(e1*e3-e0*e2);

dcos[3] =2*(e1*e2-e0*e3);
dcos[4]=e0^2-e1^2+e2^2-e3^2;
dcos[5] =2*(e2*e3+e0*e1);

dcos[6] =2*(e0*e2+e1*e3);
```

```
dcos[7]=2*(e2*e3-e0*e1);
dcos[8]=e0^2-e1^2-e2^2+e3^2;

** perform the matrix multiplication
g_body[0]=dcos[0]*g_x + dcos[1]*g_y + dcos[2]*g_z;
g_body[1]=dcos[3]*g_x + dcos[4]*g_y + dcos[5]*g_z;
g_body[2]=dcos[6]*g_x + dcos[7]*g_y + dcos[8]*g_z;

** pass the array

g_out = passArray(g_body);
}


** If the dialog data is inconsistent for simulation, abort.
on checkdata
{

}


** Initialize any simulation variables.
on initsim
{

** Assume body coordinates line up with inertial coords at
** initialization.

e0 = 1.0;
e1 = 0.0;
e2 = 0.0;
e3 = 0.0;

** allocate the arrays to be passed.
makearray(omega, 3);
makearray(dcos, 9);
makearray(g_body, 3);
}


** User clicked the dialog HELP button.
on help
{
showHelp();
}
```

Scc Merger

This block will merge the input from two semi-circular canals by weighing them according to direction of rotation:

Output = topGain*topInput
+ botGain*botInput;

with appropriate sign dependence

Written by Nick Groleau 6/92

```
** scc merger */
** totalSteps, curStep, totalTime, curTime, deltaTime */
** are defined by the system */

real                    g1, g2, s1, s2;
real                    m, p;

on checkdata
{
if ((con1in && novalue(topgain))     ** if connected and no gain entered,
                || (con2in && novalue(botgain)))     ** abort!!
        abort;
}

on initsim
{
        g1 = topGain;
        g2 = botGain;

        if (novalue(g1))
                g1 = 0.0;
        if (novalue(g2))
                g2 = 0.0;
}

** simulation part */
on simulate
        {

        s1 = Con1In/realabs(Con1In);
        if (Con1In == 0)
                s1 = 0;
        s2 = Con2In/realabs(Con2In);
        if (Con2In == 0)
                s2 = 0;

        m = (g1 -g2)/2;
        p = (g1 + g2)/2;

        Con4Out = Con1In*(p+m*s1);
        Con5Out = Con2In*(p-m*s2);
```

```
        Con3Out = Con4Out + Con5Out;
        }

on help
{
showHelp();
}
```

Square Root

This block takes 1 number and returns the square root of the number.

Written by Nick Groleau September 1990

```
** square root*/
** totalSteps, curStep, totalTime, curTime, deltaTime */
** are defined by the system */



on checkdata
{

}

on initsim
{

}

** simulation part */
on simulate
        {

        SqrtOut=sqrt(NumberIn);

        }


on help
{
showHelp();
}
```

Vector Amplifier

This amplifier is a summing amp, and will add the vectors from the input connectors times their respective gains:

```
Output = topGain*topVector
                + midGain*midVector
                + botGain*botVector;
```

Written by Dan Merfeld October 1989
(Modified from ImagineThat amplifier block)
Modified by Dan Merfeld 7/21/90

```
** vector amplifier */
** totalSteps, curStep, totalTime, curTime, deltaTime */
** are defined by the system */

real                    g1, g2, g3;
real                    tempOut[];
real                    temp1In[],temp2In[],temp3In[];

on dBs
{
** check for neg values */
if (not novalue(topGain) && topGain < 0.0)
        {
        topinvert = TRUE;                       ** changes checkbox */
        topGain = -topGain;   ** corrects gain */
        }
if (not novalue(midGain) && midGain < 0.0)
        {
        midinvert = TRUE;
        midGain = -midGain;
        }
if (not novalue(botGain) && botGain < 0.0)
        {
        botinvert = TRUE;
        botGain = -botGain;
        }
** convert vals to dBs */
topGain = 20.0*log10(topGain);
midGain = 20.0*log10(midGain);
botGain = 20.0*log10(botGain);
}

on value
{
** convert to value */
topGain = 10.0^(topGain/20.0);
midGain = 10.0^(midGain/20.0);
botGain = 10.0^(botGain/20.0);
}
```

```
on checkdata
{
if ((Vector1In && novalue(topgain))          ** if connected and no gain entered,
                || (Vector2In && novalue(midgain))          ** abort!!
                || (Vector3In && novalue(botgain)))
        abort;
}

on initsim
{
makearray(tempOut,3);
makearray(temp1In,3);
makearray(temp2In,3);
makearray(temp3In,3);

if (dBs)        ** if in dBs only */
        {
        ** convert sim vals to values for simulation */
        g1 = 10.0^(topGain/20.0);
        g2 = 10.0^(midGain/20.0);
        g3 = 10.0^(botGain/20.0);
        }
else
        {
        g1 = topGain;
        g2 = midGain;
        g3 = botGain;
        }
if (topInvert)          ** is it inverted? */
        g1 = -g1;
if (midInvert)
        g2 = -g2;
if (botInvert)
        g3 = -g3;
if (novalue(g1))
        g1 = 0.0;
if (novalue(g2))
        g2 = 0.0;
if (novalue(g3))
        g3 = 0.0;
}

** simulation part */
on simulate
        {
        GetPassedArray(Vector1In,temp1In);
        GetPassedArray(Vector2In,temp2In);
        GetPassedArray(Vector3In,temp3In);

        tempOut[0] = temp1In[0]*g1+temp2In[0]*g2+temp3In[0]*g3;
        tempOut[1] = temp1In[1]*g1+temp2In[1]*g2+temp3In[1]*g3;
        tempOut[2] = temp1In[2]*g1+temp2In[2]*g2+temp3In[2]*g3;

        VectorOut = passArray(tempOut);
```

```
        }

on help
{
showHelp();
}
```

Vector Differentiator

The differentiator outputs the first order backward difference of a three element vector:

out = gain*(newInVector-oldInVector)/deltaTime;

Written by Dan Merfeld October 1990
(Modified from Imagine That scalar differentiator)
Modified by Dan Merfeld 7/21/90

```
real tempIn[];
real tempOut[];
real oldIn[3];

** This message occurs for each step in the simulation.
on simulate
{
** call getPassedArray
GetPassedArray(NewIn,tempIn);

** perform the calculations three times
tempOut[0] = gain*(tempIn[0]-oldIn[0])/deltaTime;
oldIn[0] = tempIn[0];
tempOut[1] = gain*(tempIn[1]-oldIn[1])/deltaTime;
oldIn[1] = tempIn[1];
tempOut[2] = gain*(tempIn[2]-oldIn[2])/deltaTime;
oldIn[2] = tempIn[2];

** pass the array
ConOut = passArray(tempOut);
}


** If the dialog data is inconsistent for simulation, abort.
on checkdata
{
if (noValue(gain))
        abort;
}


** Initialize any simulation variables.
on initsim
{
makearray(tempIn,3);
makearray(tempOut,3);
oldIn[0] = 0.0;                    ** initial values are zero
oldIn[1] = 0.0;
oldIn[2] = 0.0;
}

on createBlock
{
```

```
gain = 1.0;      ** initial value
}

** User clicked the dialog HELP button.
on help
{
showHelp();
}
```

Vector Integrator

Gain is in voltsOut per volt-second in. The integrator output voltage is set to the initial condition at the start of simulation. You can select between the Euler and Trapezoidal methods. The Trapezoidal method is more accurate, but has more delay. This block calculates the integral of a three component vector input.

Written by Dan Merfeld October 1990
(Modified from ImagineThat scalar integrator)
Modified by Dan Merfeld 7/21/90

```
real    a[4],b[4],c[4];
real    tempIn[];
real    tempOut[];

on createmodule
{
init1 = 0.0;
init2 = 0.0;
init3 = 0.0;
Gain = 1.0;
}

on checkdata
{
if (novalue(init1+gain))
        abort;
if (novalue(init2+gain))
        abort;
        if (novalue(init3+gain))
        abort;}

on initsim
{
makearray(tempIn,3);
makearray(tempOut,3);

integrateInit(a, init1/gain);
integrateInit(b, init2/gain);
integrateInit(c, init3/gain);}

on simulate
{
GetPassedArray(VectorIn,tempIn);

if (euler)
{
        tempOut[0] = gain*integrateEuler(a, tempIn[0], deltaTime);
        tempOut[1] = gain*integrateEuler(b, tempIn[1], deltaTime);
        tempOut[2] = gain*integrateEuler(c, tempIn[2], deltaTime);
}
else
{       tempOut[0] = gain*integrateTrap(a, tempIn[0], deltaTime);
```

```
        tempOut[1] = gain*integrateTrap(b, tempIn[1], deltaTime);
        tempOut[2] = gain*integrateTrap(c, tempIn[2], deltaTime);
}

VectorOut = passArray(tempOut);
}

on help
{
showHelp();
}
```

Vector to Components

This block accepts a three element vector as its input. It then breaks this vector into its three scalar components.

Written by Dan Merfeld October 1989
Modified by Dan Merfeld 7/21/90

```
real dummy [];
real g_xout,g_yout,g_zout;

** This message occurs for each step in the simulation.
on simulate
{

** call get PassedArray with the connector and the array that it
** will be assigned to.

GetPassedArray(CosineIn,dummy);

** break out the three components
gx_out = dummy[0];
gy_out = dummy[1];
gz_out = dummy[2];

}


** If the dialog data is inconsistent for simulation, abort.
on checkdata
{

}


** Initialize any simulation variables.
on initsim

{

makearray(dummy, 3);

}


** User clicked the dialog HELP button.
on help
{
showHelp();
}
```

Matrix Multiply

This block accepts the error vectors as inputs. It multiplies these error vectors by the feeback matrix to yield the error input to the internal model.

Written by Dan Merfeld 7/21/90

```
real s[],g[];
real g_out[],s_out[];

** This message occurs for each step in the simulation.
on simulate
{

GetPassedArray(s_error_in,s);
GetPassedArray(g_error_in,g);

** perform the matrix multiplication
s_out[0]=k11*s[0]+k12*s[1]+k13*s[2]+k14*g[0]+k15*g[1]+k16*g[2];
s_out[1]=k21*s[0]+k22*s[1]+k23*s[2]+k24*g[0]+k25*g[1]+k26*g[2];
s_out[2]=k31*s[0]+k32*s[1]+k33*s[2]+k34*g[0]+k35*g[1]+k36*g[2];
g_out[0]=k41*s[0]+k42*s[1]+k43*s[2]+k44*g[0]+k45*g[1]+k46*g[2];
g_out[1]=k51*s[0]+k52*s[1]+k53*s[2]+k54*g[0]+k55*g[1]+k56*g[2];
g_out[2]=k61*s[0]+k62*s[1]+k63*s[2]+k64*g[0]+k65*g[1]+k66*g[2];

** pass the array
s_error_out = passArray(s_out);
g_error_out = passArray(g_out);
}


** If the dialog data is inconsistent for simulation, abort.
on checkdata
{

}


** Initialize any simulation variables.
on initsim
{

** allocate the arrays to be passed.
makearray(s, 3);
makearray(g, 3);

makearray(s_out, 3);
makearray(g_out, 3);
}


** User clicked the dialog HELP button.
on help
```

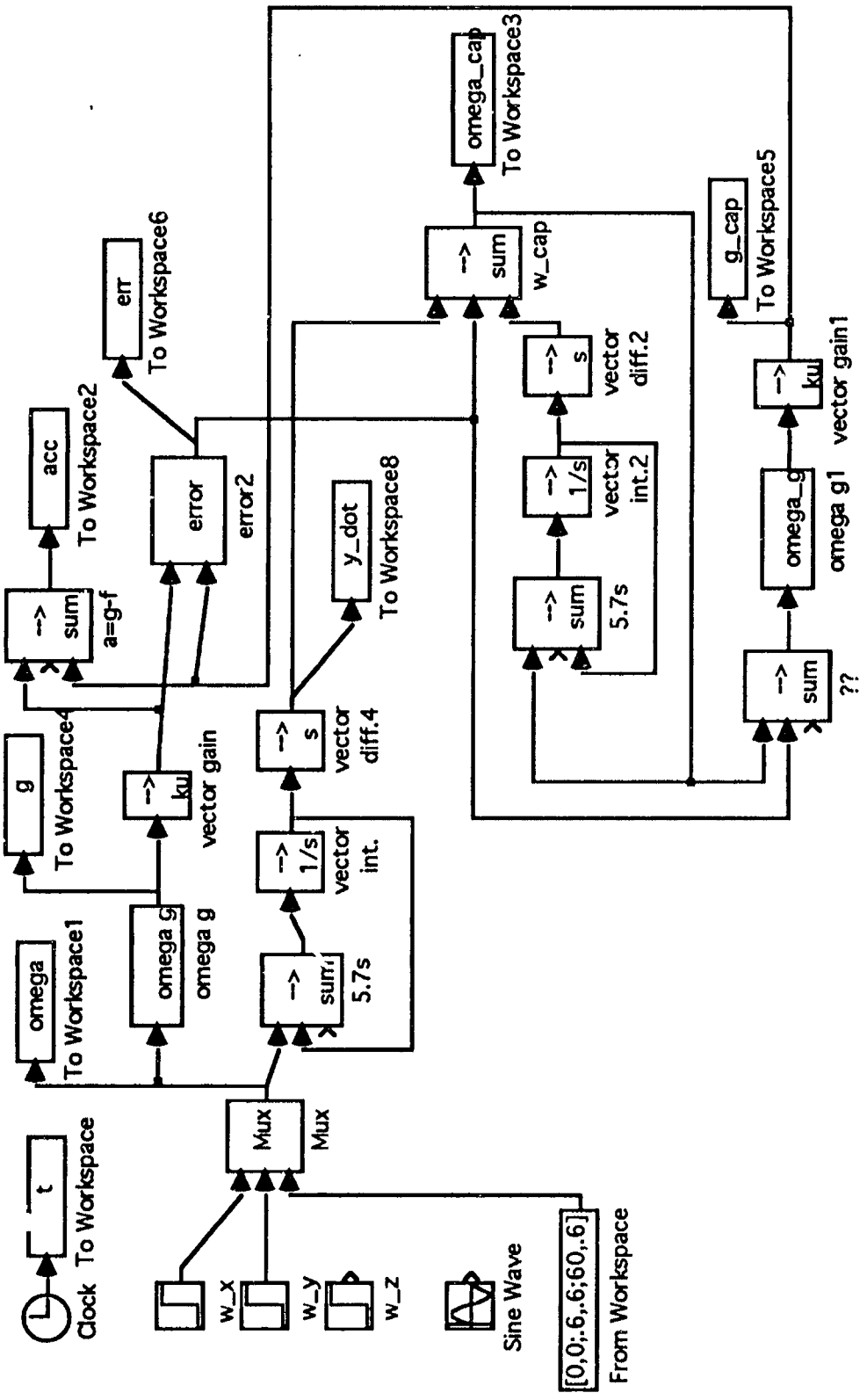
APPENDIX F

SIMULAB Modeling Code

This code implements the vestibular model and all the necessary sub-components in MATLAB/SIMULAB 1.1. The model is somewhat difficult to read but rather efficient.

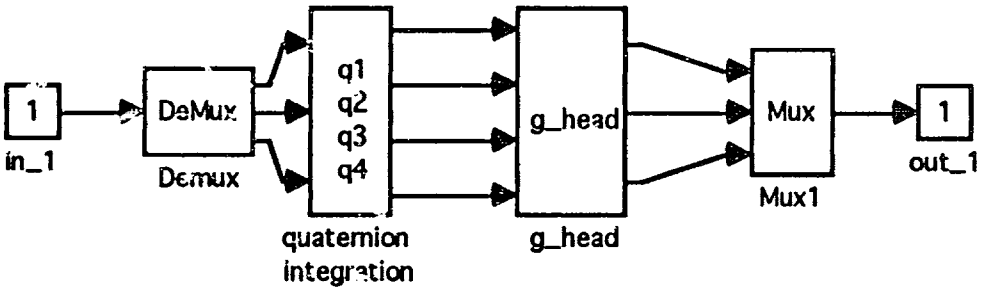
Whole Model

Written by Nick Groleau 4/92



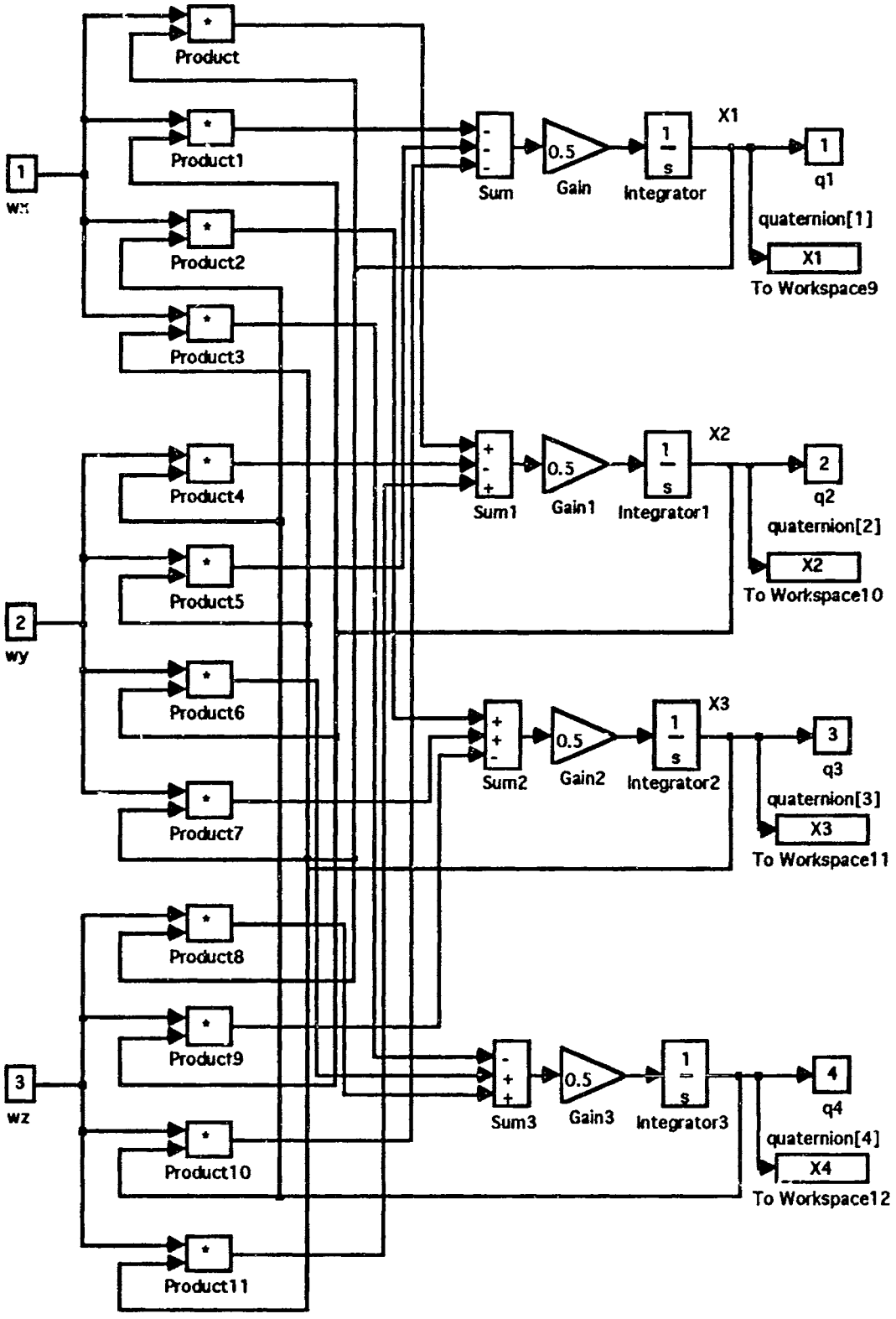
Omega G

Written by Nick Groleau 4/92



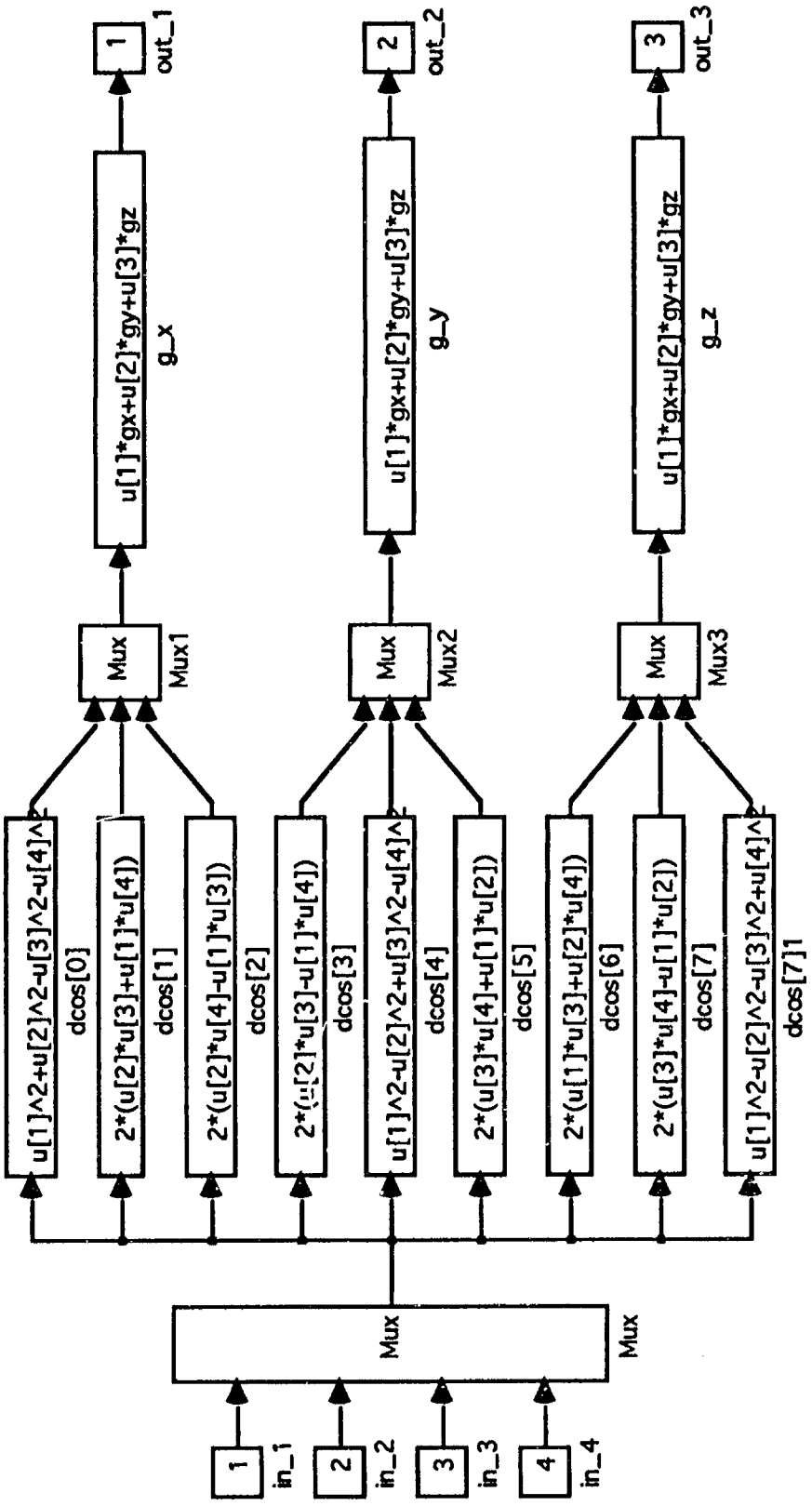
Quaternion Integration

Written by Valérie Bilien 2/92



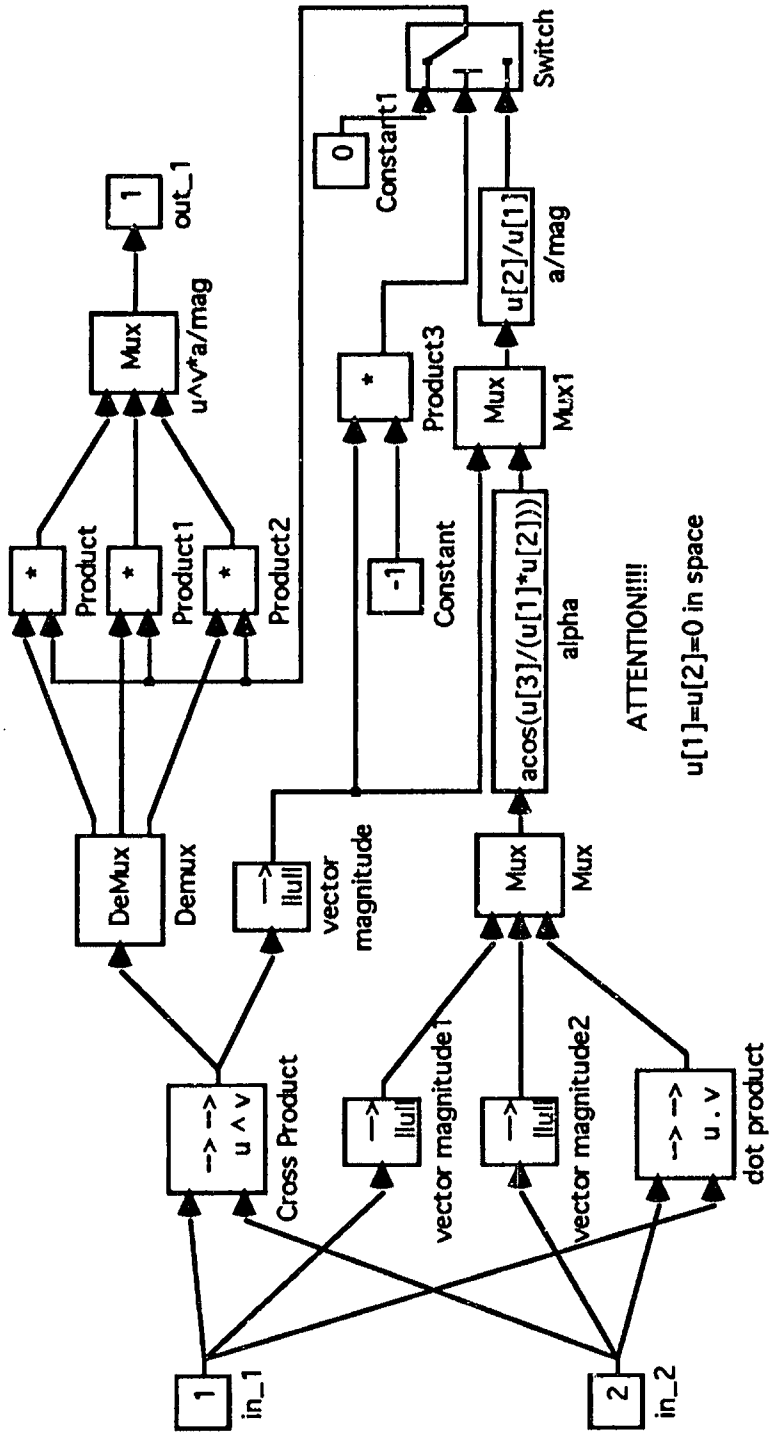
G Head

Written by Nick Groleau 4/92



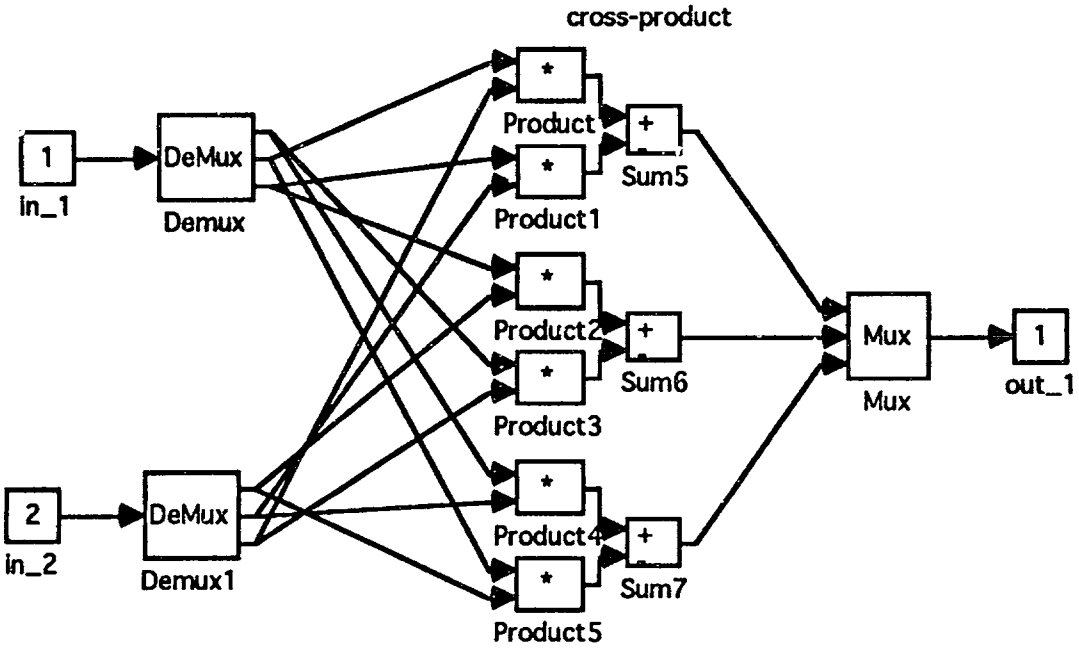
Error

Written by Nick Groleau 4/92



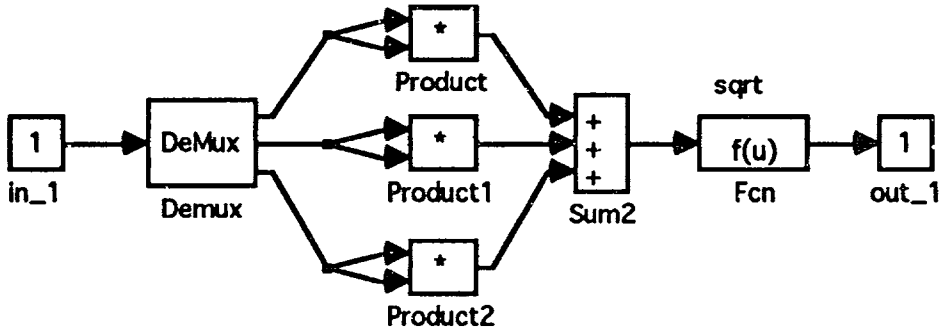
Cross Product

Written by Nick Groleau 4/92



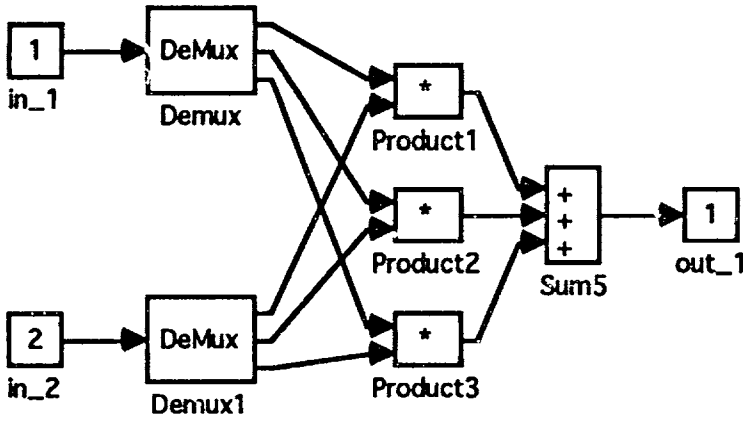
Vector Magnitude

Written by Nick Groleau 4/92



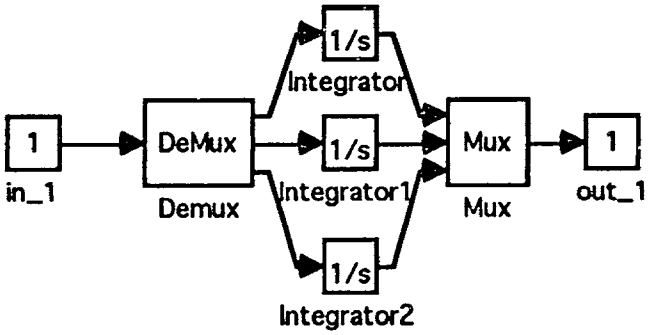
Dot Product

Written by Nick Groleau 4/92



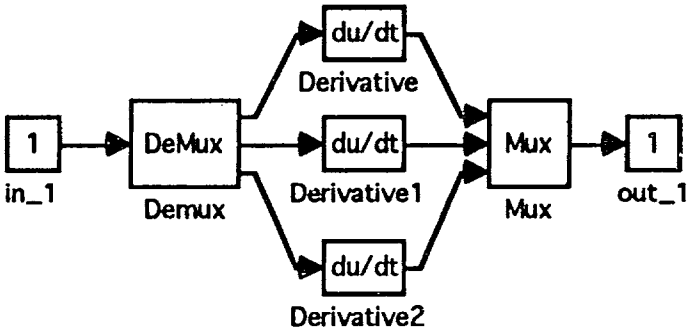
Vector Integrator

Written by Nick Groleau 4/92



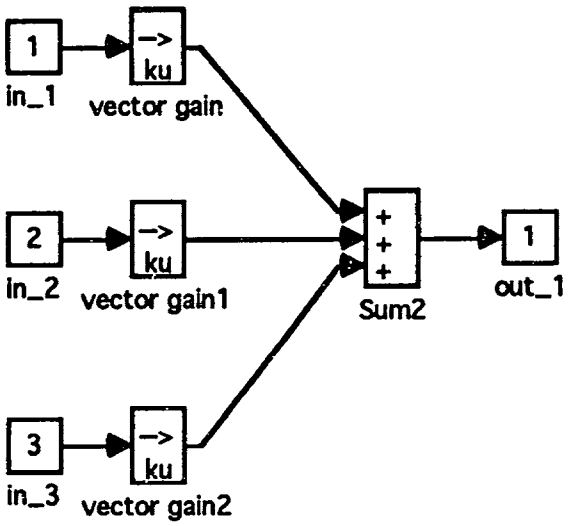
Vector Differentiator

Written by Nick Groleau 4/92



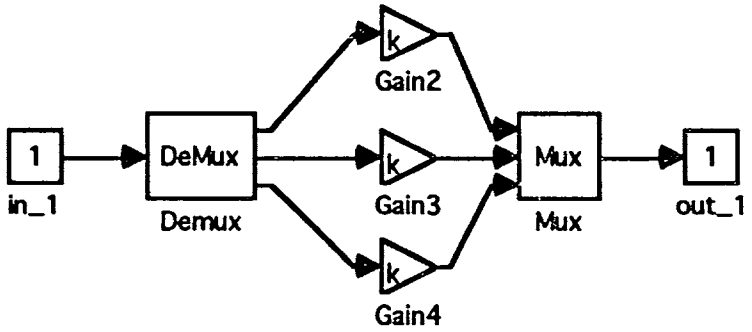
Vector Sum

Written by Nick Groleau 4/92



Vector Gain

Written by Nick Groleau 4/92

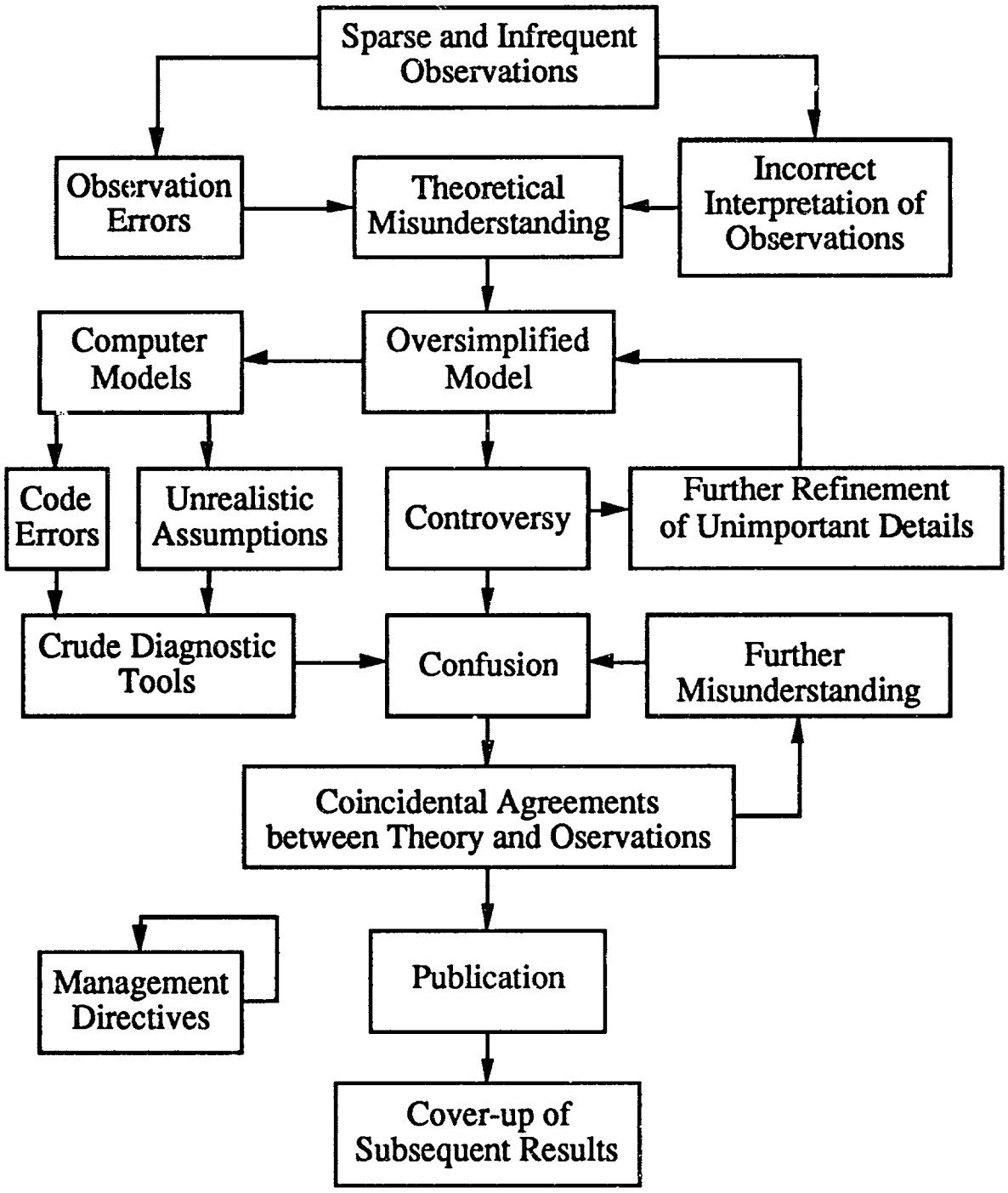


APPENDIX G

The Course of Science: A Lighter Side

This diagram illustrates what many of us are doing but were afraid to ask.

From an Office Door in Buidling E10

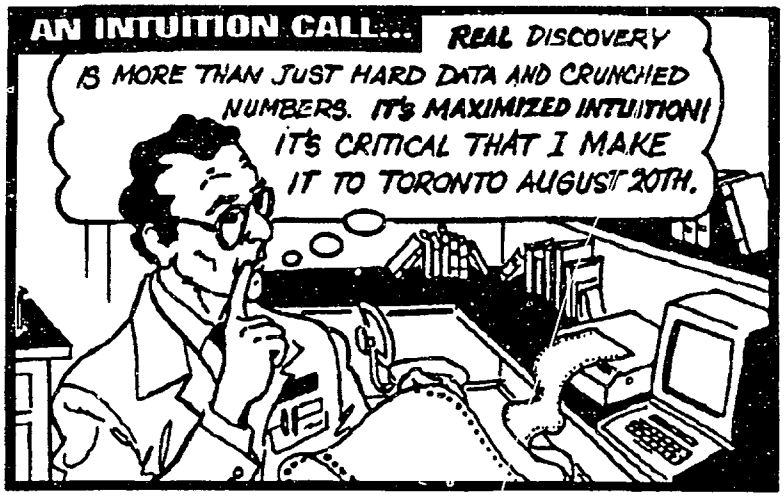


APPENDIX H

Real Discovery is Maximized Intuition

I found this ad on an airplane back from Houston and found it so appropriate that I spent some time making my own version of it.

Scientific American, June 1992, page 131



Is it before and after...

Your intuitive management skills can be understood, taught, maximized for increased profitability, production and efficiency.

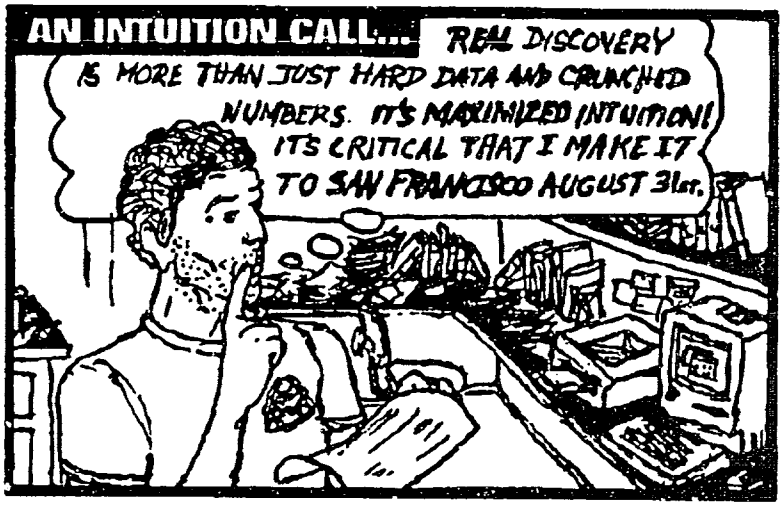
Discover how at the 2nd Annual Global Intuition Network International Conference on intelligence and intuition in Toronto August 20-22.

Write or fax today for information about intuitive management skills, free Network membership or the Toronto Conference.

Global Intuition Network
ATTN: Dr. Weston H. Agor
Box 614
University of Texas-El Paso
El Paso, Texas 79968
FAX (915) 747-5111

Global Intuition Network

or after and before?



Your intuitive derangement skills can be understood, taught, maximized for increased laughability, disillusion and entertainment.

Discover how at the Once Only Local Intuition Network Transnational Conference on intelligence and intuition in San Francisco August 31.

Write or fax today for information about intuitive derangement skills, free Network membership or San Francisco Conference.

Local Intuition Network
ATTN: Dr. Nicolas Gruisou
Box P192
University of Nemoro
Palo-Alto, CA 90556
FAX (415) 555-5111

Local Intuition Network