Giving Directions to Computers via Two-handed Gesture, Speech, and Gaze

by

Edward Joseph Herranz

S.B. Mathematics with Computer Science
Massachusetts Institute of Technology
Cambridge, Mass.
1990

SUBMITTED TO THE MEDIA ARTS AND SCIENCES SECTION,
SCHOOL OF ARCHITECTURE AND PLANNING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July 1992

Signature of Author ______________________________
Media Arts and Sciences Section
July 2, 1992

Certified by ______________________________
Richard A. Bolt
Senior Research Scientist, MIT Media Laboratory
Thesis Supervisor

Accepted by ______________________________
Stephen A. Benton
Chairman, Departmental Committee on Graduate Students

Room 14-0551
77 Massachusetts Avenue
Cambridge, MA 02139
Ph: 617.253.2800
Email: docs@mit.edu
http://libraries.mit.edu/docs

DISCLAIMER NOTICE

Giving Directions to Computers via Two-handed Gesture, Speech, and Gaze

by

Edward Joseph Herranz

Submitted to the Media Arts and Sciences Section,
School of Architecture and Planning
on July 2, 1992, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Previous work on free-hand manual gesture input to computers has focused on one-handed input used in a "direct control" or "direct manipulation" manner. This thesis explores the *synergistic* effect resulting from the concerted action of two hands in free-hand gestural input, with concurrent speech and gaze. The spirit of the exchange is one of delegation or indirect manipulation: describing how something is to be done to the machine as "agent," rather than using computer "tools" to do a task oneself in a hands-on, "direct manipulation" sense. The prototype simulation is one of arranging three-dimensional objects. The basic interactions consist of scalings, rotations and translations. This research represents a step toward eventual *multi-modal natural language communication* with machines. A video has been submitted along with this thesis, showing a sample interaction-session with the system.

Thesis Supervisor: Richard A. Bolt
Title: Senior Research Scientist, MIT Media Laboratory

The author acknowledges the support of the Defense Advanced Research Projects Agency under Rome Laboratories Contract F30602-89-C-0022.

Thesis Reading Committee

Read by ....

Kenneth W. Haase
Assistant Professor of Media Arts and Sciences
MIT Media Laboratory

Read by ....

Patricia Maes
Assistant Professor of Media Arts and Sciences
MIT Media Laboratory

Contents

List of Figures

Acknowledgments

I wish to thank Dr. Richard Bolt for his supervision of the work, as well as his creative suggestions. Thanks also to my readers, Dr. Kenneth Haase and Dr. Pattie Maes, for their comments.

This work belongs just as much to the entire *Advanced Human Interface Group* as it does to me: I wish to personally thank my friend David Berger for helping me debug my code, for answering my math questions at all times of the day or night, and for figuring out how to convert HP bitmaps into PostScript, all at the expense of his own time. Thanks to Carlton Sparrell for his unselfish willingness to share his knowledge of electronics and computer graphics, as well as some of his code. Thanks to Chris Wren and Michael Johnson for developing *Mverse*. Infinite gratitude to Kris Thorisson for editing the video which accompanies this thesis, as well as volunteering his music for the soundtrack. Thanks to David Koons for his always-useful suggestions; many of the ideas here were inspired by him. Thanks to Masaru Sugai for his "rotation routines" which he sent from Japan.

I thank Lena Davis for helping me get my money back. Thanks to Trevor Darrell and Tanweer Kabir from *Vision Modelling* for teaching me how to include figures with psfig. I thank Linda Peterson for her assistance in dealing with all of MIT's red tape.

Thanks to the *Interactive Cinema Group* for lending me their video cameras and their editing facilities. I thank Professor David Zeltzer of the *Computer Graphics and Animation Group* for lending us his DataGloves when we were in need. Thanks to the Media Lab's *Systems Group*, for answering all sorts of questions.

I thank all of my friends for encouragement when things weren't going well. I wish to thank my parents for helping me whenever I could not make ends meet.

And finally, I thank the USENET for valuable hints.

Notices

The following trademarks are referred to throughout this document:

- IBM AT is a registered trademark of the International Business Machines Corporation.
- DataGlove is a registered trademark of VPL Research, Inc.
- DragonDictate is a registered trademark of Dragon Systems, Inc.
- RTI, TURBOSRX and Starbase are registered trademarks of Hewlett Packard.
- RK-426 is a registered trademark of ISCAN, Inc.
- 3Space is a registered trademark of Polhemus Navigation Sciences.

Chapter 1

Introduction

One of the principal goals of the Advanced Human Interface Group (AHIG) at MIT's Media Lab is to make it possible for people to communicate with computers through concurrent speech, gesture and gaze. Refer to [1] for a discussion of some of the issues that this research group is involved in. Not everyone can use a keyboard or a mouse, or program a computer, but most of us commonly use speech, gestures and gaze in order to communicate. The current input technologies allow: eye-tracking, gesture-tracking and discrete speech recognition. The modes for machine output are: computer graphics and speech synthesis. The integration of such technologies is a task far from trivial, and there are many complex issues to deal with. The technologies used are often not extremely dependable, so their imperfections have to be taken into account, and sometimes some hardware has to be built.

The AHIG has developed a system which allows a user to communicate to a machine via concurrent two-handed gestures, speech and gaze (see Figure 1). The interaction environment deals with the arrangement of three-dimensional figures. The interactions consist of scalings, rotations and translations. This environment will allow to demonstrate the types of desired communication, with a stress on two-handed gestures. Gazes consist mainly of fixations. Fixations are used to determine reference, a form of "visual deixis." The gestures studied consist of a variety of two-handed actions. The types of gestures allowed are *spatiographic* and *kinemimic*.

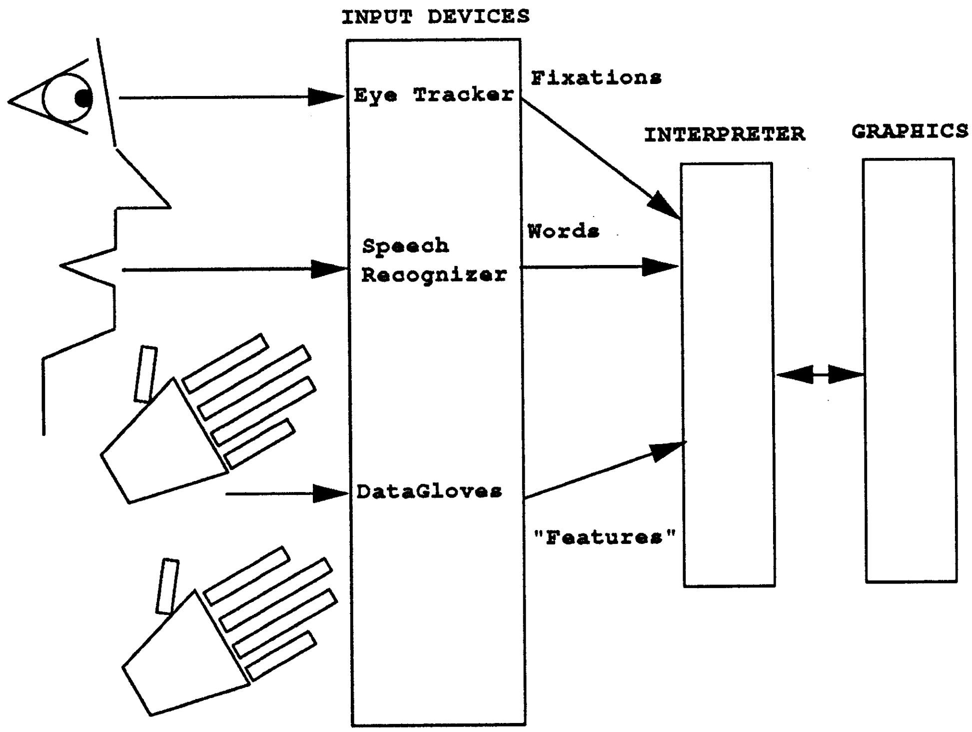

Figure 1-1: Diagram of Overall System

Refer to Section 1.1 for more details.

1.1 Historical Perspective

An ancestor of this work was "Put That There," [1] a system developed by the former Architecture Machine Group at MIT, which used speech and a hand-held three dimensional pointer as the modes of interaction [2]. Another precursor was Starker's *Gaze-directed Graphics World with Synthesized Narration*, which allowed a user to obtain information about objects located on a revolving planet by simply looking at them [31]. The current work combines all three modes of input– speech, gesture and gaze– with gestures expanded to two handed actions, not simply pointing.

Most of the prior work done in gesture recognition by computer has dealt with one-handed

[1] See section 2.5.2 for a more detailed discussion on this project.

direct manipulation interfaces. Examples of this work are Weimer's and Ganapathy's synthetic visual environment [37] and work by Sturman et al on interaction with virtual environments [33]. The latter work classifies interacting with virtual environments using hand-gestures into three modes. The first mode consists of the user "reaching" into the simulation to manipulate elements of the graphical world. In the second mode, the gesture-trackers can be thought of as abstracted graphical input devices such as a button or a locator. In the third mode, hand movements can be thought of as a stream of tokens in some language, e.g., the American Sign Language. The current work is not concerned with these three modes of interaction, but rather with having the computer "listen" and "look" at the user and "understand" him or her.

In the present study, interaction between human and machine occurs outside of the graphical environment. The interaction is one of delegation or indirect manipulation: describing how something is to be done to the machine as "agent" rather than using computer "tools" to perform a "direct manipulation."

This work views computers as "agents" that help humans accomplish tasks. The sense of the interaction is not that of "interface as mimesis," in the way described by Laurel in [19], where the computer interface becomes "transparent" to the user and the user "performs" all the tasks by himself or herself; instead, the user's requests are performed by the "agent."

The computer should be able to understand our "gestural language"; the user shouldn't have to learn a "new language" to interact with a computer. The types of gestures that will be analyzed will be deictic, spatiographic and kinemimic, which are all coverbal gestures.

Nespoulous references a gestural typology in [26] that includes the type of *coverbal* gestures, which most closely describes the kinds of gestures that will be studied in this work. Coverbal gestures can be illustrative, expressive or paraverbal. Illustrative gestures can be deictic, spatiographic, kinemimic, or pictomimic. Deictic gestures consist of pointing at objects which are the referent of a lexical item found in the subject's speech. Spatiographic gestures consist of the outlining of the spatial configuration of the referent of one of the lexical items. Kinemimic gestures outline the action referred to by one of the lexical items. Finally, pictomimic gestures outline some formal properties of the referent of one of the lexical items. The other types of coverbal gestures–expressive and paraverbal gestures–will not be covered in this current study. Expressive gestures are facial expressions and arm/hand movements to convey emotion. Paraverbal gestures are head or hand movements accompanying speech intonation and stress.

1.2 Thesis Objectives

The performed work consists of building a system that allows a user to interact with a graphical environment using two-handed gestures, speech and gazes. People communicate with each other via speech gesture and gaze; this work will attempt to allow users to communicate with a machine in a similar fashion.

The chosen context of interaction is one such that all the discussed concurrent modes of human-computer communication can be demonstrated extensively. The context consists of a three-dimensional graphical world with objects. The user can refer to the objects via fixations. Different kinemimic gestures can be used to modify the location of objects.

Although this section may give the impression that the integration of the system is a herculean task, it should be noted that a couple of similar systems, which have concurrent speech, gestures and gaze, have already been implemented. So most of the modules described in this section already exist. The novelty of this thesis is that it is the first time that **two-handed** gestures are incorporated with speech and gaze.

1.3 System Components

1.3.1 Abstraction of Device Input

There should be at least one level of abstraction above the raw data from the input devices, otherwise it would be hard to make any sense out of the continuous stream of raw information that "pours" from the input devices. For speech the lowest level is words. Then, sentences are formed which translate into actions. For eyes, the raw data is translated into fixations. For gestures, this abstraction process is more complex for many reasons. Several different techniques are under study by our group. One consists of looking at the velocity plots of the positions of the hands to locate points of interest. This approach was used in one of the prototypes of the system.

1.3.2 Eye-tracking

The current eye-tracking method, code and hardware, used for this work is as documented by Thorisson [35]. The current eye-tracking hardware is a head-mounted system, giving the user a considerable amount of freedom of movement. The code associated with the eye-tracker performs calibration and interpolation of data, as well as detection of fixations.

1.3.3 Gesture-tracking

The existing software to drive the DataGloves was used. As discussed recently in Section 1.3.1, some work is being done now to detect "gesture features." Refer to Appendix 3 for details on the hardware.

1.3.4 Hardware to Sync Eye-tracker and Two DataGloves

Digital circuitry was designed and built in order to allow the head mounted eye-tracker to function in unison with the two DataGloves. All three devices use `Polhemus` magnetic sensing cubes as their spatial locators. These cubes need to be synchronized in order to work properly in unison, without interference.

In order to be able to operate both DataGloves and the head-mounted eye-tracker all at once, the maximum update frequency of any of these devices has to be reduced to 20 Hz. This is a considerable limitation, but did not prevent the ideas presented here from being realizable.

1.3.5 Parser

A natural language parser was developed in C. The parser itself is not very sophisticated; there was no real need to build a powerful parser because the sentence format is very limited. Refer to Section 4.7 for details.

1.3.6 Coordinating Multi-modal Input

All of the information from the input devices has to be coordinated somehow and interpreted together as a whole. This was to be done in a way similar to that described and implemented by David Koons [18]. All the three input modes, speech, gesture and gaze are individually time stamped. Then this multi-modal information is included in a *frame*. *Frames* are then analyzed to see if a significant action by the user took place.

1.3.7 Input Interpreter and Graphics World

Different modules have been implemented that will piece together all of the information obtained from the different modes of communication and perform the desired action on the objects in the graphical world. Mverse, an object-oriented graphical manager was developed by the AHIG for general purposes. The "interpretation" code was the most complex part of the system, as well as the hardest to implement.

1.4 Justification

It is natural for humans to interact with other humans using speech, glances and gestures. Thus, it is be useful to assess whether such type of interaction between human and computer is feasible with today's available technology. The context of transforming objects in space suggests itself because of its richness of interaction possibilities.

The interface implemented in this thesis is powerful. It would be very hard to emulate it with today's "standard" interface tools, such as the keyboard, the mouse, and knobs. Interacting with objects in space is complicated; if one were to "link" graphical transformations to such things as an array of knobs, the resulting interface would be non-intuitive and large. For example, if a given knob translates an object along its horizontal axis in "object space," [2] then the direction of movement which is viewed depends on the object's orientation; Turning the knob clockwise does not guarantee the same result, sometimes the object may move to the right, other times to the left, depending on which way the object is facing.

1.5 Relevance of Gaze, Gesture and Speech

In order to justify this thesis it is necessary to show that the communication modes that will be used have proven to be useful. It must be shown that these modes convey information when used in human-human conversation and that the current technology is capable of obtaining accurate enough data from these modes. The three communication modes used in this work are: gaze, gesture and speech.

1.5.1 Gaze

Moving the eyes is natural; it requires little conscious effort. When a human is interested in a nearby object he or she tends to look at it. The retina of the eye is not uniform. The central part of the retina, the *retina*, is the only section that allows sharp vision. Eye movements can be broadly classified into fixations and saccades. Saccades consist of rapid motion to locate the *fovea* in a different portion of the visual scene. During fixations the eye does not remain still, several types of small, jittery motions occur [15]. The eyes are always active, so all eye-movements cannot be interpreted as intentional.

The AHIG has used different eye-tracking setups. These setups have all employed corneal/pupil

[2] See Section 4.9.1.

reflection technology. They require that a camera is focused on one of the user's eyes, and that an infra-red light source is aimed at the same eye. Refer to Appendix 3 for details. The first eye-tracker used immobilized the user's head by forcing him or her to use a chin rest. The second one had servo-driven mirrors attached to the camera, allowing the system to follow the user's pupil without the need of a head restraining device. This latter tracking method, developed as a Master's thesis by Tim McConnell [21], did not prove to be accurate enough. The current eye-tracking system consists of a small head mounted camera, with an infrared LED light source, along with a Polhemus locating-cube, to sense the position of the user's head in space.

India Starker developed a system in which a user could interact with a graphical environment with the use of gazes; refer to [31] for details. The system would communicate information to the user about objects selected by the user's gaze.

1.5.2 Gesture

Gestures are bodily movements that are considered expressive of thought or feeling. They add information to speech. Gestures are natural forms of communication between humans. Due to hardware limitations, only hand gestures were analyzed in this work. Gestures are a manifestation of a spontaneous mode of representation of meaning, and such representations can become standardized and transformed into arbitrary symbolic forms (American Sign Language) [25]. Arbitrary gestures are not considered in this work. Nespoulous and Lecours give a taxonomy of gestures in [26]. Gestures can be one of three different types: arbitrary, mimetic or deictic gestures.

Humans often use *both* their hands when gesturing; therefore both hands ought to be tracked. Buxton and Myers show that certain tasks, representative of those found in CAD and office information systems, can be significantly improved by the use of two hands as opposed to one [5]. Some previous work using two-handed gestures and speech in the context of narrative episodes has been done at the AHIG [3]. In this work, a simple example of an episode of vehicles in motion was implemented, which also allowed the user to spatially lay out static objects with the use of gesture. Please refer to Appendix 3 for details on the hardware used for gesture tracking.

1.5.3 Speech

Speech is the most powerful communication channel, but it is often insufficient alone. In human-to-human conversation, many details are left out because they are part of an "obvious" context, which could be picked up if the computer knew where the individuals were glancing at, and how they were gesturing, what they were pointing at, and so on. This does not imply that the computer would be able to pick up the entire context of the conversation, but it would do significantly better than if all it could do was listen to the conversation.

It is common for humans to describe spatial relationships to each other, and aid the description with hand gestures. For example, everybody has experienced being given directions by someone who uses his hands to describe the path we should take. It almost seems that one could not give directions without using hand gestures, for they are a very "natural" mode of communication to use.

Chapter 2

Background

2.1 Justification of Gestural Interfaces

Originally, our hands were nothing but pincers used to hold stones: Man's genius has been to turn them into the daily more sophisticated servants of his thoughts as a homo faber and as a homo sapiens.—André Leroi-Gourhan, Gesture and Speech, 1964–1965 [1]

Hauptmann conducted an experiment with people using gestures and speech to manipulate graphic images on a computer screen in 1989 [13]. A human was used as the "recognition device." The analysis showed that people strongly prefer to use *both* gestures and speech for the graphics manipulation, and that they intuitively use multiple hands and multiple fingers in all three dimensions. A surprising amount of uniformity and simplicity in the gestures and speech was discovered. Hauptmann concluded that the evaluation of his results strongly indicates the development of integrated multi-modal interaction systems.

Users were asked to perform three types of operations: *rotation, transposition,* and *scaling*

[1] As quoted by Nespoulous and Lecours in [26]

operations. The users averaged the use of 1.2 hands for rotation, 1.1 hands for transposition and 1.5 hands for scaling. Out of all the gestures used by the 36 subjects who participated in the experiment, 199 gestures were classified as rotations, 137 as translations and 128 as scaling gestures. 58% of the subjects preferred the use of gesture and speech, 19.4% preferred gestures alone and 22.2% liked speech alone the best.

In 1986, Buxton and Myers conducted two experiments to study two-handed input [5]. The experimental tasks were representative of those existing in CAD and office information systems. The first experiment consisted in evaluating the performance of a compound selection/positioning task. The two subtasks were performed by different hands using separate transducers. Novice users quickly learned to accomplish both tasks simultaneously, without any suggestion. The speed at which the task was accomplished was directly related to the degree of parallelism which the test subject used. This experiment showed that users are able to cope with using both hands at once to their advantage in the right environment.

Their second experiment consisted of evaluating the performance of a compound navigation/selection task. It compared a one-handed versus a two-handed method for selecting words in a document. People who used both hands did significantly better than those using only one, but very few of the subjects used both hands simultaneously (the ones that did had the fastest times of all).

2.2 Gestural Interfaces: Introduction

Gestures are defined by *Webster's VII Dictionary* as the use of motions of the limbs or body as a means of expression. They are not random components of the communication process; they have varying degrees of specification. Gesture is a fundamental part of direct interpersonal-communication, just as speech and facial expressions.

Not everyone can use a keyboard or a mouse, or program a computer, but most of us use speech, gestures and gazes in order to communicate. It seems important to develop technology to allow computers to recognize these modes of communication in order to investigate the potentials of this type of interface. None of these interpersonal-communication modes is independent of each other but, obviously, speech is the most powerful one, when considered alone.

This section is a study of gestures and their potential use as an interface to a computer. Gestures are not considered alone, though; they are analyzed together with speech. Gestures usually "fill-in" elements omitted in speech, but they can also convey full meaning, independent of speech. For example, one might ask someone "Which of these objects do you prefer: the blue one or the red one?" to which he responds by merely pointing at the red object.

This is a review of current gesture taxonomies and the work up-to-date toward building computer interfaces which use gestures. This section also proposes future work in the field of gestural interfaces.

2.3 Gestural Taxonomy

Where ignorance exists theories abound.—Leucippus, ca. 460 B.C. [2]

There doesn't seem to be a widely agreed taxonomy of gestures; however, if one closely analyzes the different taxonomies in the literature, many similarities occur. Bernard Rimé and Loris Schiaratura have recently published a gestural taxonomy which collects and organizes most significant previous ones which have appeared in the literature [28]. These authors present what they refer to as the *revised Efron system of speech-related hand gestures* which is divided into three main categories: *"ideational" gestures*, *depictive gestures*, and *evocative gestures.* Efron's original system proposed in 1941, which was in part based by a classification of gestures introduced by Wundt (1900/1973) [39], contains virtually all gestures proposed in all the taxonomies thereafter.

[2] As quoted by Mary Ritchie Key in [16]

2.3.1 Gestures Referring to the Ideational Process

Gestures referring to the ideational process mark the speaker's pauses, stresses and voice intonations. According to Efron, there are two major subclasses in this category: speech-marking hand movements, and ideographs.

Speech-marking hand movements comprise what Efron named *batonlike gestures*; these time the stages of the referential activity. There are a number of gestures referred to in the literature which are a variant of *baton* movements. Freedman (1972) refers to *punctuating movements* which relate to the emphatic components of speech and *minor qualifiers* which are stylized accentuation movements [12]. McNeill and Levy (1982) distinguished *batonic movements* which stress some linguistic item that the speaker wishes to emphasize [22]. Ekman and Friesen (1972) defined *batons* as accenting a particular word or phrase. McNeill (1987) mentioned small uniform movements, which he called *beats*. Such movements appear with clauses that are performing an extranarrative role, such as anticipating a story. Cosnier (1982) gathers under *paraverbal* gestures those that stress speech emphasis, or mark the major stages of reasoning [8].

Efron considered a second class referring to the ideational process which he named *ideographs*. They consist of gestures which sketch in space the logical track followed by the speaker's thinking. Similarly, McNeill & Levy(1982) recognized what they called *metaphoric gestures*, which convey some abstract meaning occurring in speech [22].

2.3.2 Gestures Referring to the Object: Depictive Kinds

There are two types of gestures in this class, according to Efron: *physiographic* and pantomimic gestures. Physiographic gestures can also be named *iconic gestures*.

Iconic gestures consist of hand movements which parallel the speech by providing a figural representation of the object being referred to. Freedman (1972) included this type of gestures under the label of *motor primacy representational movements* [12], McNeill & Levy (1982) named them *iconic hand gestures* [22] and Cosnier (1982) mentioned them in his class of *illustrative* *gestures* [8].

The class of iconic gestures is subdivided into three categories: *pictographic*, *spatiographic*, and *kinetographic gestures.* Pictographic gestures outline some formal property of the referent of one of the lexical items. For example, an upward spiraling movement of the finger of person may be used to signify a spiral staircase. Spatiographic gestures outline the spatial configuration referred to by one of the lexical items. For instance, a speaker may use his hands to indicate the relative location of buildings or streets, when giving directions to someone. Kinetographic gestures outline the action referred to by one of the lexical items. These kind of gestures correspond to such movements of the hand as a descending motion which parallels the use of the expression "falling down."

The other category of depictive gestures proposed by Efron are *pantomimic* gestures, which consist of true mimetic actions. For instance, to illustrate the words "he grasped the hammer," the speaker's hands shape an imaginary hammer. Pantomimes often engage the whole body, so the speaker also becomes an actor. Pantomimes, in their strongest form, may not require any speech whatsoever, becoming autonomous.

2.3.3 Gestures Referring to the Object: Evocative Kinds

Efron's last two classes of gestures can be put under the same category, according to their evocative aspect. These gestures no longer depict the referent, but rather, simply evoke it. The two types of gesture in this category are *deictic* and *symbolic gestures.*

Deictic gestures are hand gestures which refer to objects that are visually or symbolically present. In lay terms, they are referred to as "pointing" gestures. This type of gesture has been considered by every gesture classification scheme. Pointing gestures can be relatively easily interpreted by a computer, with the appropriate hand-sensing hardware, and become a powerful interface tool, as will be discussed later.

Symbolic gestures, named *emblems* by Efron, are gestural representations which do not have a morphological relationship with the object that is being represented. A typical example is the waving of the hand to signify a greeting. According to Ekman and Friesen (1972), they are verbal acts that (1) can be directly translated into one or two words, (2) have a precise meaning to a group of people, and (3) are always used intentionally for the purpose of communicating a particular message to the receiver.

2.4 Direct Versus Indirect Manipulation

Some authors consider other types of gestures besides the ones referred to in Section 2.3. These kinds of gestures are what I refer to as direct manipulation gestures. For example, Buxton [6] refers to a range of gestures that are almost universal, and lists a few:

- *pointing* at objects
- *touching* objects
- *activating* objects such as controls, for example by pushing, pulling, or twisting
- *moving* the position or orientation of objects
- *mutating* the shape of objects by squeezing or stretching
- *handing* objects to others

The only one of these gestures that can be found in the previously discussed taxonomy consists of pointing gestures, more properly named deictic gestures.

Direct manipulation gestures are an interesting and useful mode of interaction; they are especially well suited for applications were a great need of precision is needed, such as computer aided design, but they are not true gestures. There is nothing to be interpreted about turning a dial, it is simply turned a certain amount and that is all there is to it. This thesis is concerned with gestures that require interpretation, not with direct manipulation interfaces.

The ultimate "direct manipulation metaphor" is perhaps best described by Brenda Laurel in [19]. She believes that the computer interface should become "transparent" [3] to the user.

In Sturman's doctoral dissertation [34], the mapping of the hand to various complex objects, such as a crane, is studied in a "direct manipulation" sense.

2.5 Systems with Gestural Interfaces

2.5.1 Hand-Sensing Hardware

The rigorous study and application of gestures has not been possible until relatively recently, simply because there were not any devices which allowed a somewhat reliable and not too obtrusive "measurement" of hand gestures. Perhaps the most widely hand gesture tracking device in use currently are the VPL DataGloves, though other devices are currently out there in the market. It is important to note that these tracking devices alone do not recognize gestures in the sense discussed in Section 2.3 by themselves, they only provide information about the hand's shape and location in space (See Section 3.1 for details).

Several Polhemus cubes can be used in unison, but the maximum sampling rate becomes $60/n$, if there are n cubes. This would be necessary, for example, if one were interested in tracking the position of the entire upper body. A new company, Ascension Technology, has developed a similar sensor, using different magnetic technology, which allows to use several cubes at once, e.g., five, in parallel without diminishing the sampling rate, which is 100 Hz. The problem then becomes dealing with too much input. Another company, Exos, makes extremely accurate hand and wrist-shape trackers, but they are considerably cumbersome.

There are other ways of doing body tracking: The Architecture Machine Group at MIT, developed a full-body optical tracking suit [4] in the early 1980's. This suit also included a glove, perhaps the first hand-tracking device: it used infrared LED's (light-emitting diodes) and a

[3] See Section 1.1.

pair of cameras with infrared filters to do the positioning. VPL also has a full-body suit, which employs the same fiber-optic technology as the DataGlove.

2.5.2 "Put That There"

Perhaps the first system developed which made use of hand gestures was "Put That There," designed with Bolt developed by the MIT Architecture Machine Group in 1979 [2]. This work allowed a user to command simple shapes about a large-screen display surface with the use of deictic gestures and speech. Pointing was detected using Polhemus sensing-technology. This project allowed the free usage of pronouns, since voice can be augmented with simultaneous pointing.

Users could create, move, change, name, interrogate and delete shapes via the concurrent use of speech and deixis. The level of accuracy of the speech recognition hardware was about 65% [2]. Chris Schmandt and Eric Hulteen, who programmed "Put That There," did everything possible so that the user would not have to repeat words unnecessarily. Computer visionary Alan Kay, in an interview in Psychology Today, described interacting with the system as being "like dealing with a friendly, slightly deaf butler...From the standpoint of your expectations you are willing to deal with it." "Put That There" was one of the first steps, if not the first one, towards multi-modal interaction involving speech and gesture combined.

2.5.3 "Virtual Environments"

The concept of a "virtual environment" is an abstract space in which the user places himself "inside." Sturman, Zeltzer and Pieper [33] classify interacting with virtual environments using hand-gestures into three modes: The first consists of the user "reaching" into the simulation to manipulate elements of the graphical world. For the second one, the gesture-trackers can be thought of as abstracted graphical input devices such as a button or a locator. In the last mode, hand movements can be thought of as a stream of tokens in some language, e.g., the American Sign Language.

The issue of "virtual environments" distracts us from our goal of analyzing gestural interfaces. Gestures have to be interpreted whether the user places himself "inside" or "outside" of the computer. Some of the work that has been done in "virtual reality" consists of connecting a number of users into a common "virtual world" where they can interact. Users can then use gestures and body language to communicate with each other–it is the hope of the author that computers will be able to understand them as well.

2.5.4 Systems with Gesture and Speech Input

Neal et al [24], developed the CUBRICON system (Calspan-UB Research Center Intelligent CONversationalist). This system was part of the Intelligent Multi-Media Interfaces project, which is devoted to the application of artificial intelligence methods to further the development of human-computer interface technology which will integrate speech input and synthesis, natural language text, graphics and deictic gestures for interactive man-computer dialogues. CUBRICON incorporated pointing and simple drawings with speech input; a mouse was used for input, *not* the *free hand.*

At the same time, a different system was developed by the MIT Media Lab's Advanced Human Interface Group (AHIG) [4] [18]; it incorporated free-handed pointing, eye-tracking and speech. This system represents a step forward in *functionality* over "Put-That-There" in that it allows the user to indicate the referent by either hand or eye, or both.

Cohen and Sullivan et al [7] developed two systems similar to CUBRICON, also in the late 80's. SHOPTALK, an intelligent interface system in the domain of circuit manufacturing, developed at SRI ,and CHORIS, a similar system, at Lockheed. Natural language and deixis can be used to interact with these two systems. The advantage of these systems with respect to CUBRICON is that they are better equipped to deal with the difficult problems of natural language anaphora, the phenomenon of parts of speech referring back to previous parts; it can be considered as "verbal deixis." Anaphora is resolved via graphical rendering and deictic gestures.

[4] Both CUBRICON and the AHIG's multi-modal interface system were funded by DARPA and monitored by the Rome Air Development Center

Weimer and Ganapathy developed a practical synthetic visual environment for use in CAD and teleoperation in 1989, at AT&T Bell Laboratories [37]. Hand gesturing was used to implement a virtual control panel, and some three-dimensional modeling tasks. The addition of simple speech significantly improved the interface.

Also in the late 1980's Herranz implemented some of Bolt's ideas about two-handed gestures with speech [3]:

- indicating graphical manipulations (*kinemimic*)
- specifying static layouts (*spatiographic*)
- describing dynamic situations (*kinemimic*)

2.5.5 Using Neural Nets for Symbolic Gesture Detection

In 1990, Fels & Hinton, from the University of Toronto, developed Glove Talk. The system consisted of a VPL DataGlove connected to one end of a neural net, and a speech synthesizer on the other. The result is a system that can, in real time, recognize a subset of American Sign Language (ASL), and generate continuous speech (with control of stress and speed through gesture).

Murakami and Taguchi developed a similar method for recognition of Japanese sign language in 1991 [23]. A basic neural network was trained to recognize an alphabet of 42 symbols. Then, a recurrent neural network was used to detect words, which are specific combinations of the forementioned symbols.

Neural networks cannot be effectively used to detect *coverbal* gestures because they are not as well defined as sign-language (*symbolic*) gestures are. There would be infinitely many training samples. Please refer to Appendix C for details on the back propagation algorithm.

2.6 Gesture Recognition

Action is a sort of language which perhaps one time or other, may come to be taught by a kind of grammar-rules.—William Hogarth (1753), Analysis of Beauty [5]

Detecting specific gestures is not extremely complicated in sign language, as we will see in Section 2.5.5, but most other types of gestures are hard (because "templates" cannot be used). Work must be put into developing ways of abstracting three-dimensional gesture data.

2.6.1 Feature Extraction: Searching for the "Phonemes" of Gestures

Current hand-tracking technology can provide a measure of the hand's position in three-dimensional space and the hand's shape. It is very hard to detect complex types of gestures, such as *coverbal gestures*, directly from this data. Perhaps the simplest type of gestures consist of signs, that are easily recognized. Signs, typically, are inflexible gestures. For example, consider the gestures of sign language: if they were too flexible they could not function. Signs are easily recognized because data from the device can be mapped directly to the sign: a direct template matching. Murakami and Taguchi [23] developed such a system in 1989.

In order to facilitate the interpretation of complex *coverbal* gestures, there should be at least one level of abstraction beyond the raw data provided by the tracking device. This level of abstraction can be thought of as an an attempt to distinguish some characterizing components of all gestures: in other words, the *"phonemes of gestures,"* or *"gestural features."*

The following discussion on feature extraction is based on some initial work done at the AHI [6] Group of the MIT Media Lab. The gesture-tracking hardware employed by this research group consists of the VPL DataGlove (Refer to Section 2.5.1 for details). This device provides the x,y,z coordinates and euler angles (roll, pitch and yaw) of a small sensor located on the hand, and angle values of the first two joints of all the fingers. The position sensor is located on the

[5] As quoted by Mary Ritchie Key in [16]

[6] Advanced Human Interface

part opposite the palm. All of these sensors are actually attached to a flexible-cloth glove which the user wears.

If attention is paid only to the raw device data when analyzing a complex gesture, such as a *kinemimic gesture*, we are faced with a large stream of positional coordinates and finger-joint values which is very hard to make sense out of. One of the first requirements of extracting generalized features out of gestures was to somehow develop a mechanism that could divide a gesture into "strokes." David Koons, a PhD candidate at the AHI Group, suggested simply calculating the magnitudes of three dimensional velocity and acceleration vectors.

Changes in velocity and acceleration of the hand typically mark the beginning and ending of gestures, as well as characterizing the subparts of gestures. Most of the analyses done by the AHI Group have concentrated on velocity. It may prove useful to pass the magnitude of the velocity through a "filter," using the normal, or Gaussian, distribution.

Another "filter" which could be used is the inverse of the sigmoid function (the inverse of the standard threshold function used in neural networks trained with the back propagation method).

The filter smoothens inaccuracies of the hardware [7]. The filtered data also seems to provide a more accurate depiction of what actually happens because when the velocity increases the number of data records received per unit time decreases as the device sampling rate is a constant.

[7] Glitches in the data occur randomly.

Chapter 3

Hardware

This chapter describes the hardware that was used for this project.

3.1 Gloves

The DataGlove is a device, produced by VPL, Inc, which can transmit, in serial, the location, orientation and shape of a given user's hand. The location and orientation is calculated by the Polhemus cubes (P-cubes) an instrument constructed by Polhemus Navigational Sciences, Inc. The P-cubes provide three-dimensional coordinates and the three Euler angles of the sensor-cube with respect to a basis set up by the base-cube, with an accuracy of 1/60th of an inch. The P-cubes use magnetic field-sensor technology.

The shape of the hand is determined using VPL's own VPL fiber-optic technology: Optic fibers are used to measure the degree of flexion of finger-joints. Light is emitted through one end of the fiber and the amount of light received on the other end is also recorded. As a finger-joint is bent, the amount of light transmitted via the fiber decreases. The relation between the bending of the fingers and the light loss is not linear, as what would seem a good initial guess, but quadratic. After a simple calibration has been made, the DataGlove can figure out with great precision how many degrees a certain finger-joint has been flexed. In the standard configuration, each dataglove has ten optic fibers, two for the first two inner joints of each finger.

The data record sent out by the glove-box consists of sixteen fields. The first ten fields correspond to the angles of the finger joints (there are two angles for each finger). The next three fields correspond to the three-dimensional location of the Polhemus sensor attached to the glove. The last three indicate the roll, pitch and elevation of the Polhemus sensor. The glove-box can send these data records at a rate of 30 Hz or 60 Hz.

3.2 Head-mounted Eye-tracker

The eye-tracker is an instrument which measures the user's visual line of gaze (where he is looking in space). Tracking the position of the user's eye in space is not sufficient for human-computer dialogue. To illustrate the difference, suppose that the tracker detected a small vertical motion of the pupil. This could either indicate that the user's head moved, therefore, he would still be looking at pretty much the same point; or that the user's eyeball rotated with respect to his head, causing a large change in where the eye is looking.

There exist different technologies to perform visual-line-of-gaze tracking. The eye-tracking hardware used by the AHIG, an RK-426 developed by ISCAN, Inc., uses pupil/corneal reflection technology. The RK-426 requires a video camera (with an infrared filter) focused on one of the user's eyes as input, and an infrared light source aimed at the same eye [1]. The eye-tracker locates the pupil, which is conveniently the darkest spot on the field of view of the camera, and the corneal reflection of the infrared source, which is the brightest spot on the field of view.

Pupil/corneal reflection technology is based on a few approximations about the "mechanics" of the eye, and a mathematical derivation: Assuming that the user's head is relatively still, and that the exterior rotating section of the eyeball is hemispherical, there exists a one-to-one

[1] The levels of infrared radiation have been medically proven to be harmless to the user.

correspondence between two vector spaces. The first vector space consists of all the vectors formed by the position of the pupil and the corneal reflection on the hemisphere of the eye-ball. The second vector space consists of all the vectors which are a result of projecting the vectors in the first vector space onto a plane tangent to the the eye-ball hemisphere and normal to the "principal" axis of the forenamed hemisphere.

Now, from the "pupil/corneal-reflection plane," another one-to-one correspondence can be established with a plane in cartesian three-dimensional space. It is important to notice that the vectors in "pupil/corneal reflection space" have no absolute meaning, they cannot be thought of as vectors in the same cartesian space as where the user is actually gazing at.

3.2.1 Eye-tracking Calibration

In order to go from a pupil/corneal reflection vector to the user's visual line of gaze, or point that he's looking at on a plane, some form of calibration of the system has to be performed. Then, an interpolation can be performed from "pupil/corneal-reflection space" to "gaze space". One way of performing a calibration to allow such as interpolation would be to record the output by the eye-tracker when the user looks at a set of known points, one at a time, on a given plane (screen). Then, a simple linear interpolation can be performed. Since assumptions are made in order for the pupil/corneal reflection technique to work, the more points on the plane, or screen, that the user is calibrated on the more accurate the results will be.

In the current setup, with a head-mounted eye-tracker, slightly different methods must be used. One considerable method is to force the user's head still during calibration, and use the previously described method, recording also the distance of the user's head to the plane he was calibrated on. Afterwards, a vector from the eye to the "floating" precalibrated plane can be calculated, and the intersection of this vector with some other surface ahead can be computed as well. Dave Koons, a PhD candidate at the AHIG, has developed a calibration method which does not require the user to keep his head still at first.

3.3 Speech Recognizer

The AHIG currently uses a DragonDictate board, developed by Dragon Systems, Inc., running on a NorthGate 386 machine. It is a discrete speech recognition system with an 80,000 word vocabulary, that adapts itself to the user's voice.

3.4 Scheduling Workstation

Leon, a Hewlett-Packard (HP) 835 workstation is the central coordinating element of all of the previously discussed hardware. Attached to Leon is a real-time interface board (RTI) that runs on its own, whose sole purpose is to collect all data from the input devices and "time-stamp" it and buffer it so that different input modes can be compared to each other. All of the communications run using the RS-232 protocol.

The HP 835 workstation also serves as a graphics engine. It is equipped with hardware to do graphic transformations in real time. All of the code for this project runs on this machine. Work is currently being done at the AHIG to "parallelize" gesture, gaze and speech detection as much as possible: this would allow to divide the "work-load" among several networked machines.

Chapter 4

System

4.1 Overview

The final system I developed for this thesis allows a user to scale, rotate and translate figures in a graphical environment–with the use of gestures, speech and gaze. Three different prototypes were developed: The first one allowed interaction with a single object using "gestlets" code [1] to "prefilter" hand gestures. All of the processes ran on the same machine, rendering the overall system slow. The second prototype read the hand data directly from the VPL DataGlove boxes, increasing the total speed significantly. The final prototype incorporated the use of eye-tracking to enable users to interact with multiple objects, using gaze as the mode of reference.

Ideally, the final code should use the "gestlets" processes, for it only makes sense to pay attention to gesture data when it is significant. Refer to Section 4.5 for details on 'gestlets.'

[1] "Gestlets" were developed by fellow graduate student Carlton Sparrell. See Section 4.5 for details on "gestlets."

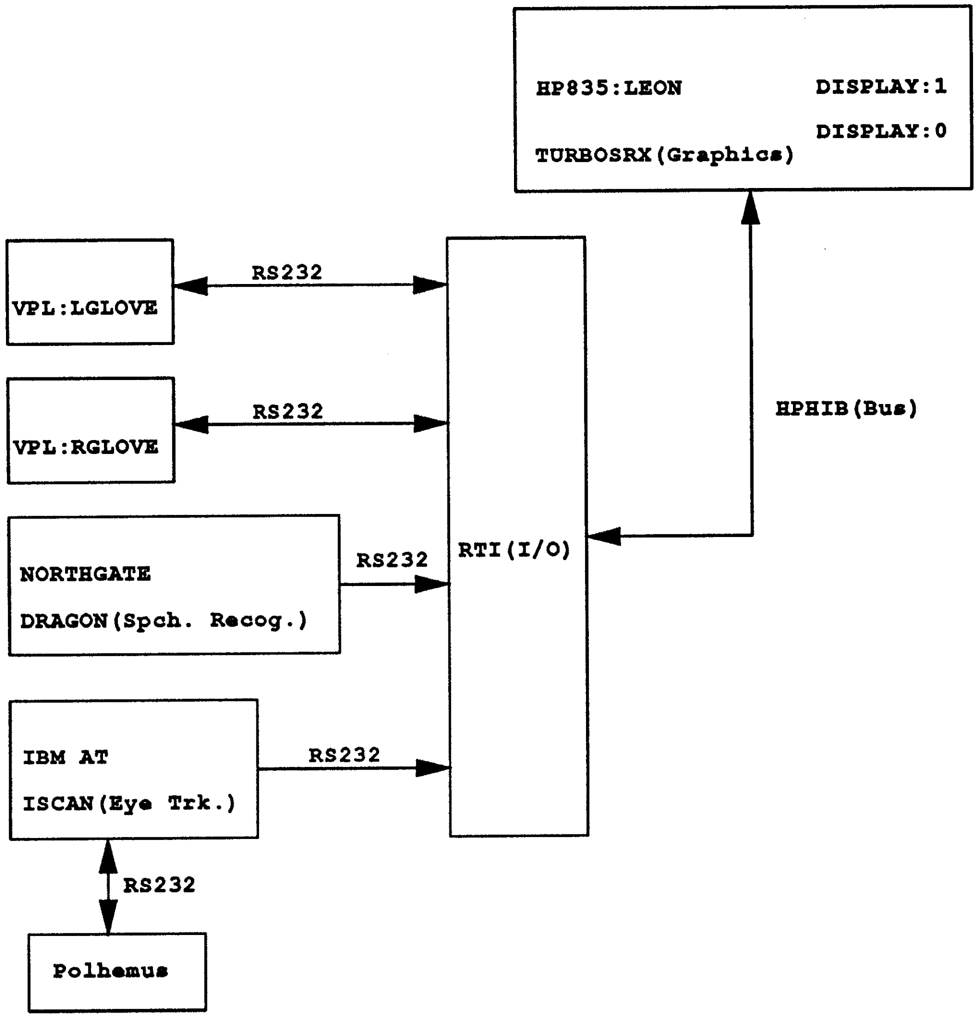

Figure 4-1: Hardware Block Diagram

4.2 Hardware

Figure 4-1 is a block diagram of the hardware used. The major components [2] are as follows: Two VPL DataGlove Boxes, a Dragon Dictate Speech Recognition System running on an IBM compatible, an ISCAN RK-426 eye-tracker running on an IBM AT, a Polhemus system connected via RS232 to the same IBM AT, and an HP835 workstation.

The VPL DataGlove boxes provide the system with the shape, position and orientation of the user's hands. The ISCAN eye-tracker provides the system with the vector between the center of

[2] Refer to Section 3 for details on these components.

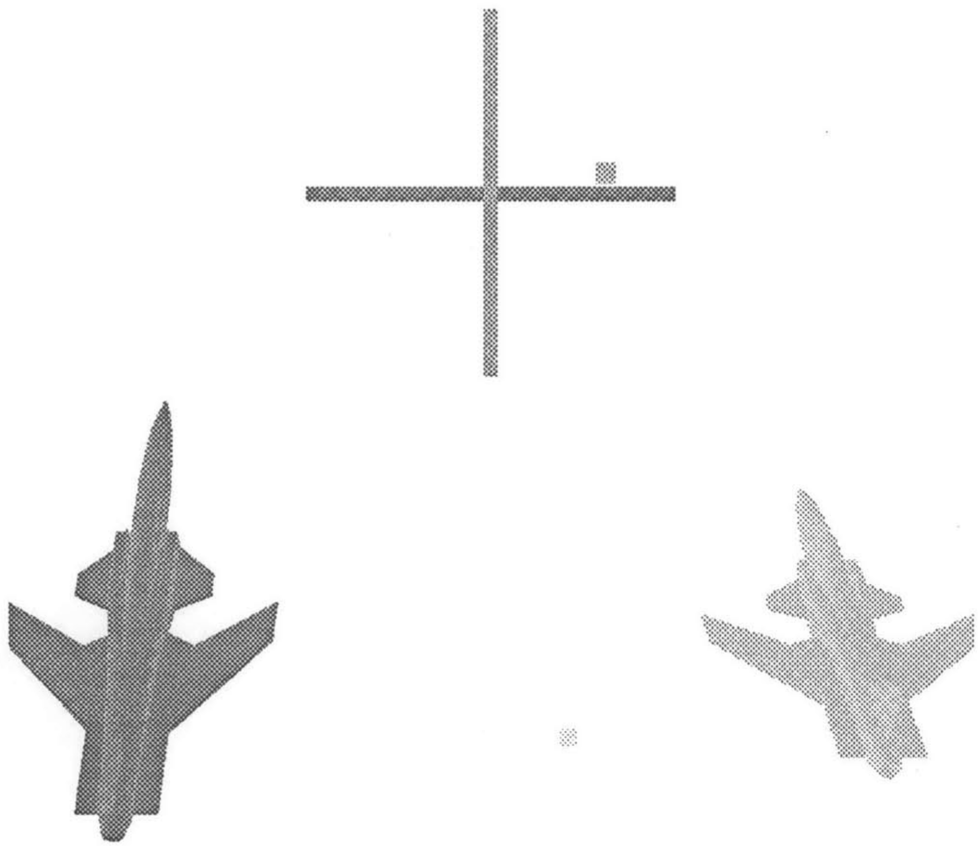

Figure 4-2: The Graphical Environment of the Prototype with Eye-Tracking: Two Jet Airplanes

the pupil and the center of the cornea reflection (Dx,Dy). Along with this vector , the position and orientation of the user's head, provided by a Polhemus system, is tagged along. From this information, it is possible to determine where on the screen the user is looking, after the eye-tracker has been successfully calibrated.

The two DataGloves and the head Polhemus must all be synchronized in order for them to function correctly. This is done by my *sync* circuitry. Refer to Section 4.3 for details.

The HP835 has a Real Time Interface (RTI) board on it which handles all input devices. This board buffers all input and "timestamps" it as it comes in for further relative comparisons. Much of the code which handles the RTI is owed to Chris Wren's [3] work. The RTI is connected to the HP835 via HP's high speed bus, the HPHIB.

The HP835 also is equipped with graphics accelerator, the TURBOSRX, which makes it a good machine for rendering images in real time, which is an essential part of this project.

4.3 Sync Circuitry

The VPL glove box provides a computer with the shape, position and orientation of a user's hand. The position and orientation values are obtained with the use of a Polhemus 3Space system [27], which is contained within the VPL hardware. Refer to Section 3 for details on their operation. The Polhemus board uses magnetic field technology to function. Not more than one Polhemus system can function at any given moment due to mutual magnetic interference (they all work on the same frequency). If it is necessary to operate more than one in unison, then it is imperative to "multiplex," or synchronize all of the systems to avoid interference.

As is suggested in the VPL Manual [36], if n Polhemus systems are to operate concurrently, then the synchronization frequency should be $\frac{60}{n}$ Hertz, to avoid noise. Thus, if we are to run three Polhemus (two DataGloves, and a Polhemus board) the sync frequency should be 20

[3] UROP Student with the Advanced Human Interface Group.

Hertz. These sync signals have to be of TTL [4] levels.

Different methods to generate a 20 Hertz pulse were considered, but the one that is about to be described was the cleanest of all. A great deal of help in the design of this circuitry came from Carlton Sparrell [5].

The VPL DataGlove boxes provide two out-of-phase 30 Hertz signals. They are intended to synchronize two DataGloves. It is not trivial to generate three out-of-phase 20 Hertz signals from a 30 Hertz one, though.

Ideally, if we had a 60 Hertz signal, it would be possible to use a universal shift register, such as an LS194, and continuously shift the sequence "100" in the same direction, after carefully connecting the correct carry-out pin to the correct carry-in pin. If we then looked at the three least significant outputs of the register, they would each be one of the desired three out-of-phase 20 Hertz signals. See Figure 4-4 for a timing diagram of these signals.

Then, it would be necessary to either develop additional circuitry to load the sequence "100" at startup time or take into account all different power-up states so that this sequence is eventually loaded, which is easier. The second approach was taken. This means that there may be a certain number of iterations right after power up in which the sync circuitry does not operate correctly, but they can be overcome. With the skillful use of a *NOR* gate, all of this was accomplished, with only a maximum number of two bad cycles, which implies that the sync circuitry is guaranteed to operate correctly in slightly over three hundredths of a second. Refer to Figure 4-3 and Table 4.1 for details.

4.3.1 Generating a Sixty Hertz Pulse

But why would it be easy to generate a 60 Hertz signal? Well, you cannot simply go into a parts store and ask for it. The traditional way is to get a clock, a crystal oscillator, of some particular speed which divides evenly into 60 by some number: $\frac{freq}{num} = 60$. Then, we can achieve

[4] Transistor-Transistor Logic

[5] Graduate student in the Advanced Human Interface Group.

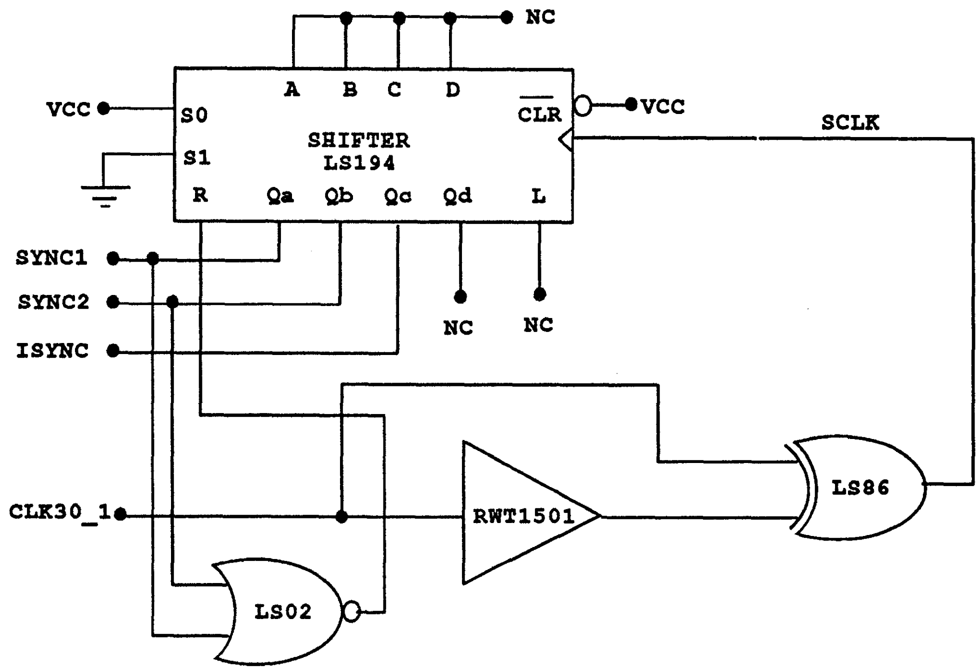

Figure 4-3: The Sync Circuit

Table 4.1: The Transition Table

$Q_aQ_bQ_c \rightarrow$	$Q_aQ_bQ_c \rightarrow$	$Q_aQ_bQ_c$
0 0 X	1 0 0	0 1 0
0 1 X	0 1 0	0 0 1
1 0 X	0 0 0	1 0 0
1 1 X	0 1 0	0 0 1

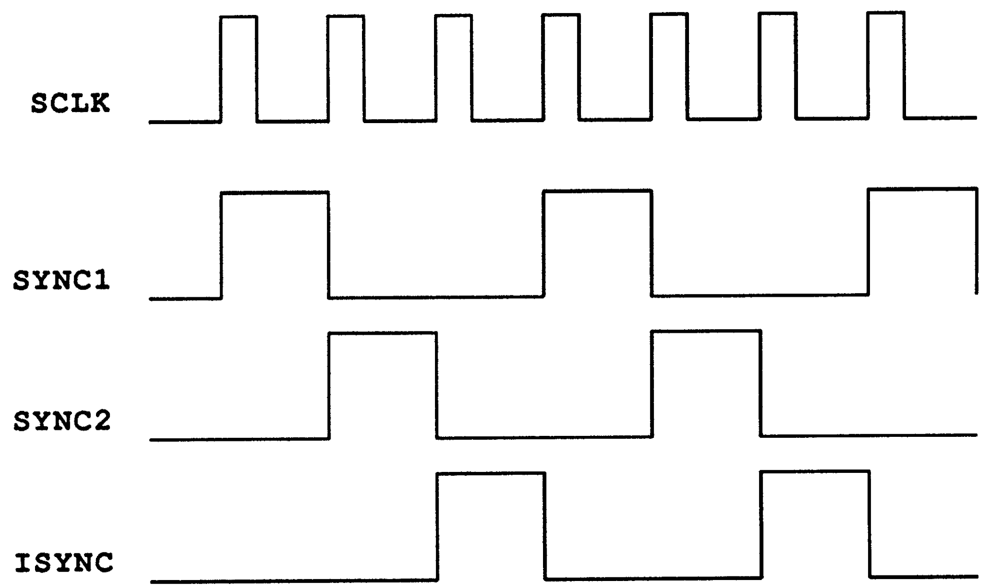

Figure 4-4: The Three 20Hz Sync Signals

the desired rate by using a certain number of decrementers in parallel (this is only because most decrementers cannot handle quantities larger than 4 bits). These decrementers load *num*, and continuously decrement the value, one by one, using the crystal's original frequency as their input-clock. When the value reaches 0, then this is a high pulse of the desired 60 Hertz signal, otherwise the 60 Hertz signal should be low.

There is a much simpler way of generating a 60 Hertz pulse: Take one of the two 30 Hertz signals provided by the DataGlove unit. Feed it as one of the inputs of a 2-input *XOR* [6] gate. Take the same 30 Hertz signal and "delay" it by some amount of time using a delay line [7]. Feed the output of the delay line as the other input of the *XOR* gate. The output of this *XOR* gate is a 60 Hertz pulse. Refer to Figures 4-3 and 4-5 for details. Using delay-lines to generate high frequency signals is generally not considered good practice, but due to the low frequency requirements of this application there is nothing wrong with this approach, especially considering that it reduces the final size of the circuit by a factor of three using standard TTL level gates.

A logic diagram of the discussed circuit can be found in Figure 4-3. A detailed diagram for

[6] A XOR B=(A AND (NOT B)) OR ((NOT A) AND B)

[7] A delay-line is a "buffer" which simply delays its input by a specific amount of time.

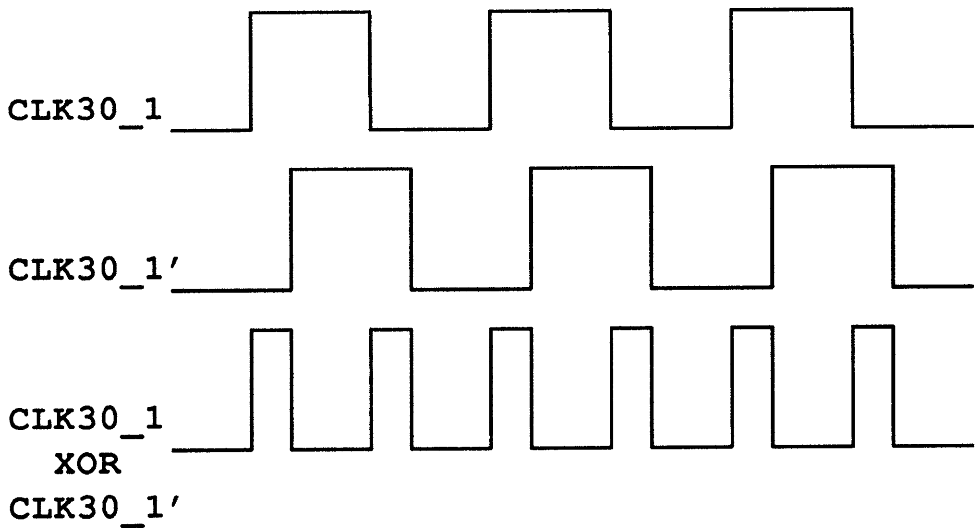

Figure 4-5: Generation of 60 Hz Clock

wiring purposes can be found in Figure 4.3.1. The size of the final circuit could have been reduced further by using a *PAL* (Programmable Logic Array) instead of the two logical gates employed, the *XOR* gate and the *NOR* gate.

4.4 Software Issues

A significant amount of software by many different programmers was used either directly or indirectly for this project. An object-oriented graphical system which runs on its own and can be communicated with via the network was used. This is *Mverse*, developed by UROP students Michael Johnson and Chris Wren, for the use of AHIG. (I thank Chris for continuously implementing my never ending additional demands of the system.) The "Gestlets" code from graduate student Carlton Sparrell was used in one of the prototypes of the one-object version. The code which reads and decodes the DataGlove information was originally written by me, and then modified for the RTI board by UROP Student Michael Johnson. The process which calculates the point of intersection of the eye gaze with the screen was developed by graduate student Kris Thorisson. Besides some borrowed libraries and functions, the rest of the code was written by me. All of these programs were developed in C. Some original work was begun

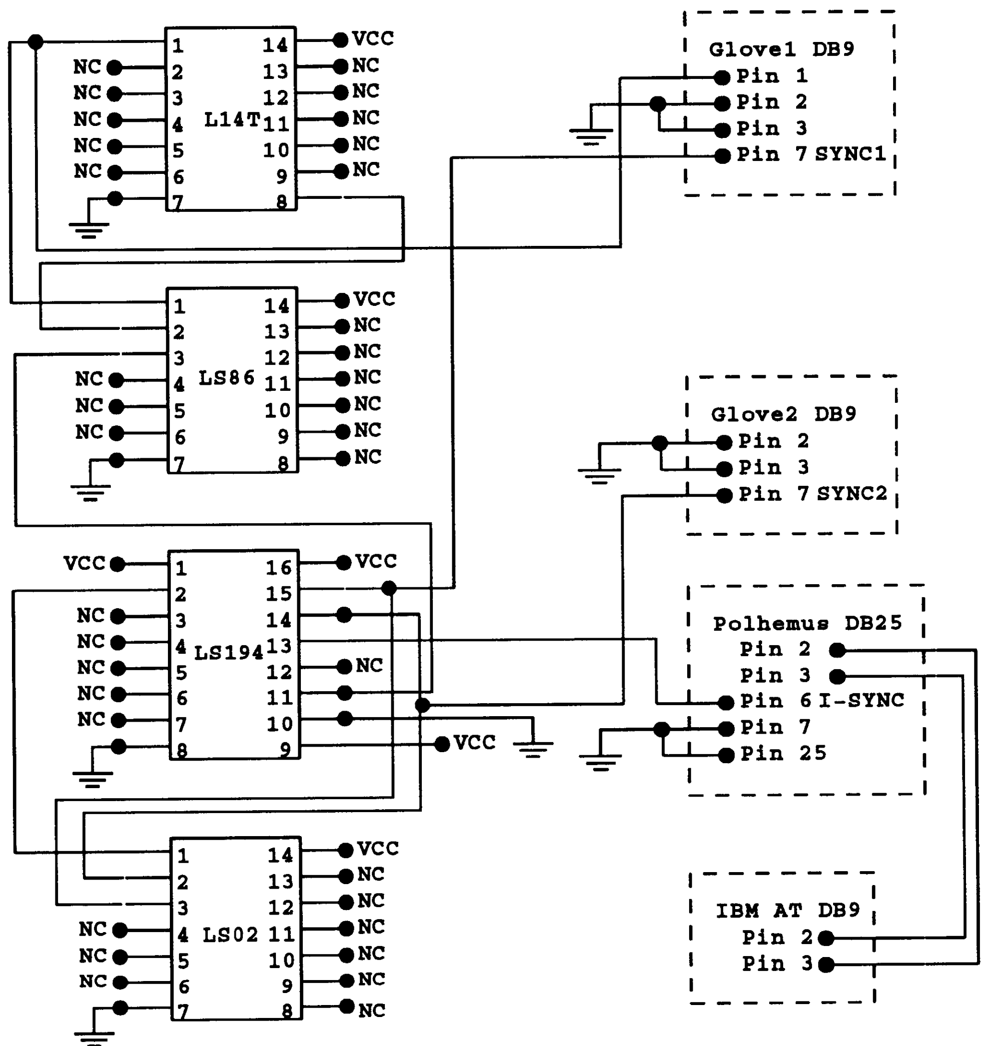

Figure 4-6: Wiring Diagram of the Sync Circuit

in Lisp, such as code for parsing sentences; but, in the interest of speed, it was all developed in C.

4.4.1 Data Flow

Figure 4-7 is an attempt to show the flow of data between the major "modules" of the project. The term "module" is in quotes because it is used loosely here: some of the "modules" consist of functions; but others are large sections of code, which can contain a significant number of functions within themselves; others are independent processes. The reason for such a loose definition is so that Figure 4-7 would be more intelligible.

The basic input to the system is speech, gesture and gaze. Speech is the driving mode of communication; that is, it is input in speech that results in some user command or request being performed. Speech is read as it becomes available on the RTI port. Gestures can be read "filtered" from a FIFO (First In First Out) file generated by the "gestlets" processes [8] or directly from the RTI port. They are read continuously, so that the information is not lost. Gazes are read from a FIFO file, which is generated by the "eyevector" process. Gazes are read at the end of a sentence, and enough space is allocated in the fifo file so it does not overflow.

The main "module" is rawwingest.c. This module initializes Mverse, calls the parser, which then in turn parses the multimodal input, interprets it, and performs the user's commands. All of the system was developed under the X windowing environment. There are a couple of reasons for this: X allowed the incorporation of a small window which echoes the user's spoken words (for debugging purposes), and it allowed a simple mechanism to poll the DataGlove records in timed intervals, using X WorkProcs.

The interactions implemented by me consist of scaling, rotation and translation requests by the user. After the gazes and gestures are decoded, together with the spoken sentence, Mverse is requested to perform the expressed transformation. The connection to Mverse is done with Berkeley sockets, allowing any machine on the net to use this graphics package.

[8] See gestlets section, 4.5

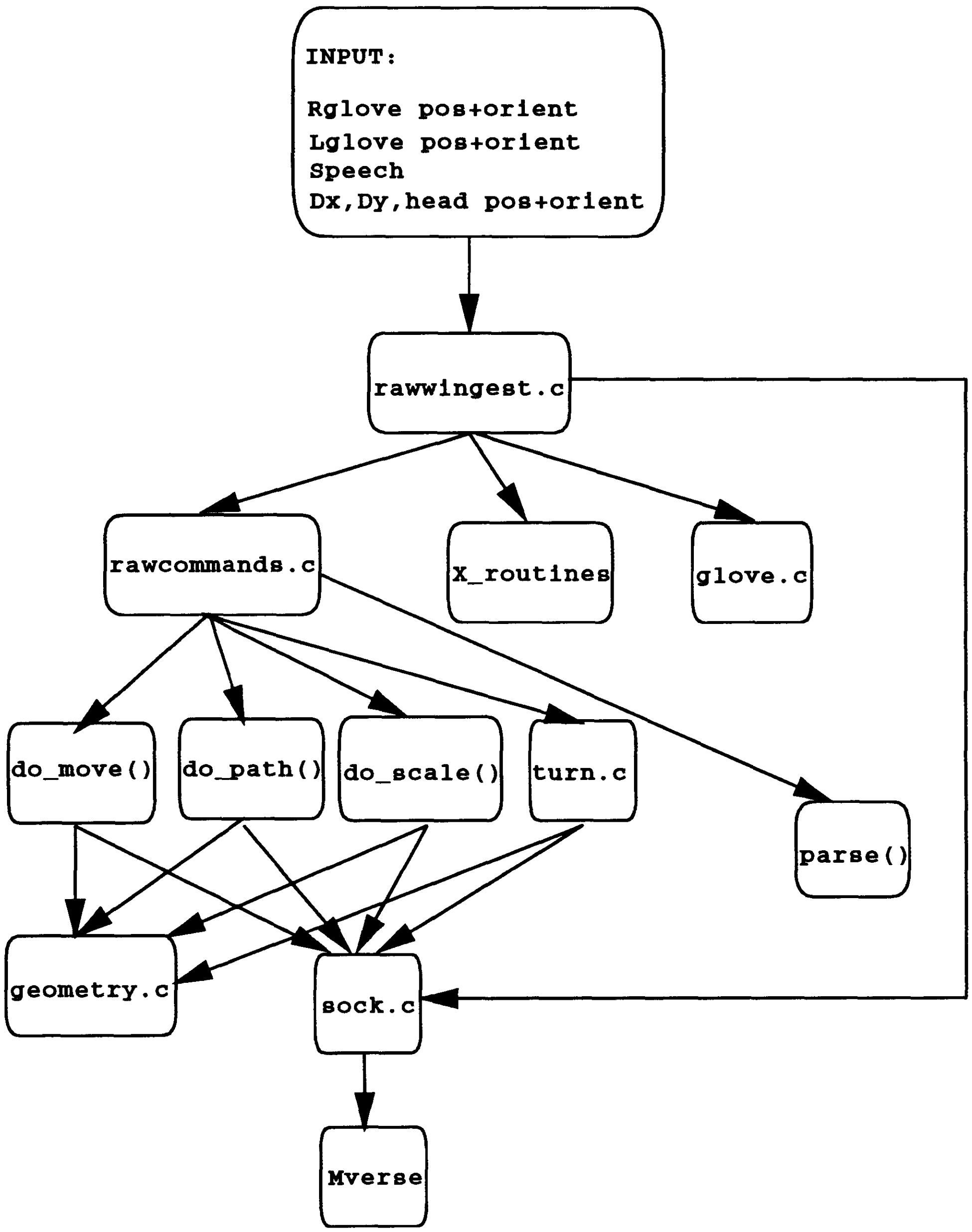

Figure 4-7: Dataflow Diagram

4.5 Gestlets

Gestlets [9] consists of a number of processes which "prefilter" the gesture data, and attempt to characterize gestures into "features." The filtering is of two types: The first filter is a "discrete" filter. If the user is not moving his hands, then there are no gestures. The velocity [10] of the gloves is analyzed and if it does not exceed a minimal threshold then no gesture is generated. The second filter attempts to clear somewhat the "noise" in the data. A Gaussian distribution of the last three samples is used [11].

Each hand has a 'gestlets' process running, which puts the 'gestlet' into a FIFO file as it occurs. At the same time there is a third process which looks at both hands' FIFO files and generates two-handed gestlets in another FIFO file. Two-handed gestlets contain an additional number of "features," describing the relationships between the gestlets of the individual hands.

"Features" consist of the following: whether the glove moved or not, which way the palm faces, whether the user pointed at the screen [12] or not and whether the hand is "relaxed" or not.

4.6 Mverse

Mverse [13] is a powerful objected-oriented network-based graphics system built on top of HP's Starbase graphics library. It allows a process running on a machine on the net to perform graphical transformations on objects.

Mverse can define, load and store graphical primitives. It was also designed with the animation of objects in mind. Animation is controlled by a world clock to allow complex interactions between multiple objects. When this thesis research project began, the goals were more numerous.

[9] Gestlets was developed by Carlton Sparrell

[10] Using velocity of gestures as the quantity to study was originally suggested by David Koons, of the AHIG, many years ago.

[11] Refer to Section 2.6.1.

[12] In order for this to be possible the screen must be calibrated in Polhemus space.

[13] Mverse was developed by UROP students Michael Johnson and Chris Wren working at the AHIG

Mverse was initially conceived to accommodate all of those goals, as well as other goals of the Advanced Human Interface Group. Unfortunately, since the accomplishment of goals always takes longer than originally planned, all of Mverse's capabilities were not used, not reflecting the hard work which the authors of the package put into developing it. For details on Mverse refer to Appendix A.

4.7 Parsing Multi-Modal Inputs

4.7.1 Time Stamping

In order to coordinate all multimodal-input, the system "time-stamps" data upon entry into its central I/O processor, the RTI (Real Time Interface) board attached to the central workstation. This board runs a special operating system, PS-OS, which is designed for I/O applications. This way, it is insured that the data is not lost and is properly buffered. It is assumed that the time which the data takes to get from the input devices themselves to the RTI board is negligible and constant.

The "timestamps" allow the system to perform time-relativistic judgements of the data when necessary, such as when "parsing" the multi-modal input. "Timestamps" are measured in hundredths of a second [14].

4.7.2 "Parsing"

The driving mode in this system is speech. No actions can be initiated without speech; all gestures and gazes get "parsed around" speech. The current prototypes look for specific words, or "tokens" (the term used in the natural language processing community), and attempt to find the gestures and gazes closest in "timestamp" to them. By and large, people tend to

[14]The "timestamp" is reset to 0 every day at midnight, thus running the system around that time of the day will produce unaccounted bugs!!

look at what they are talking about (if the item is present) and/or at whom they are talking with (again, if present). The approach I chose assumes that the correlation between tokens and gestures or gazes is parallel, but it is not necessarily clear that this always be the case in human-human interaction. We could glance at an object when trying to get someone's attention to it and then glance at the person and say "that one." At the moment we are saying "that one" we are looking at the person, but we are referring to the object. Such complicated issues in reference are being studied at the Advanced Human Interface Group by David Koons, a doctoral candidate [18].

The parse method used by this system is very simple. It can be thought of as a state-machine parser. A state-machine parser uses the current state of the sentence to predict what type of word may follow. The grammar of interaction is so simple that there are no recursive constructs, so in a sense the parsing used could be considered pattern matching at the lowest level. Refer to [30] for a detailed discussion on natural language using C. See Figure 4.7.2 for a the parse() function, which "chops" a sentence into tokens and returns the number of them. See Figure 4.7.2 for an example of a type of sentence recognized by the system.

Because parsing sentences in *multi-modal* Natural Language is a **synergistic** effect when merging gestures, gaze and speech, a relatively simple procedure for parsing speech input may serve very well.

4.8 "Natural" Transformation Gestures

The gestures which the system recognizes were chosen because they seemed "natural." Perhaps there are no gestures which are "innate" in humans [15]. "Pointing," the quintessential deictic gesture, may well be taboo or at least impolite in some cultures, and may not denote the same intentions as it does in the "Western world," but otherwise it is widely and "automatically" understood, and requires very little interpretation.

[15] Anthropologists and developmental psychologists may argue this point...

```
/* The function parse() parses a sentence and counts how many words are in
   it. It returns an array of words. A word is similar to a string, with
   '\0' in it.
*/
char **parse(sent,w_num)
char *sent; /* Sentence to be parsed */
int *w_num; /* Keeps track of the number of words in the sentence */
{
        char *word, **wrd_a, **init;
        static  char ter[]=" .?;:!"; /* List of possible termination chars */
        void *malloc();
        char *strtok();
        char *strcpy();

        wrd_a=malloc(sizeof(char *) * MAX_W);
        init=wrd_a;
        word = strtok(sent,ter);
        *w_num=0; /* The 1st word is the 0th one */
        while(word!=NULL)
        {
                /* printf("%s\n",word); */
                *wrd_a=malloc(sizeof(char) * (strlen(word)+STR_E));
                strcpy(*wrd_a,word);
                word = strtok(NULL,ter);
                wrd_a++;
                (*w_num)++;
        }
        return(init);
}
```

Figure 4-8: parse() function

```
/* Rotations */
/* "Rotate/turn that/this prism/object (like this/that)/(this way)" */

int recognize_rotation(psent,wtime,w)
char **psent;
int  *wtime,w;
{
  int matchstr();
  void do_rotate();

  if(w<5) return(0);

  if(matchstr(psent[0],"rotate") || matchstr(psent[0],"turn"))
    if(matchstr(psent[1],"this") || matchstr(psent[1],"that"))
      if(matchstr(psent[2],"object") || matchstr(psent[2],"prism"))
        if(matchstr(psent[3],"like"))
          if(matchstr(psent[4],"this") || matchstr(psent[4],"that"))
            {
              do_rotate(wtime[1],wtime[2],wtime[4]);
              return(1);
            }
          else return(0);
}
```

Figure 4-9: Sample type of sentences recognized

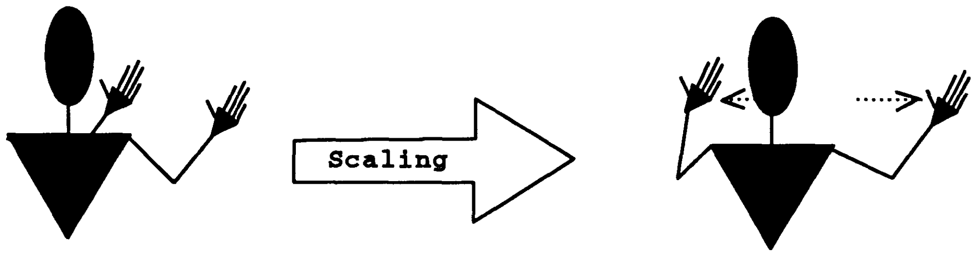

Figure 4-10: Scaling Gesture

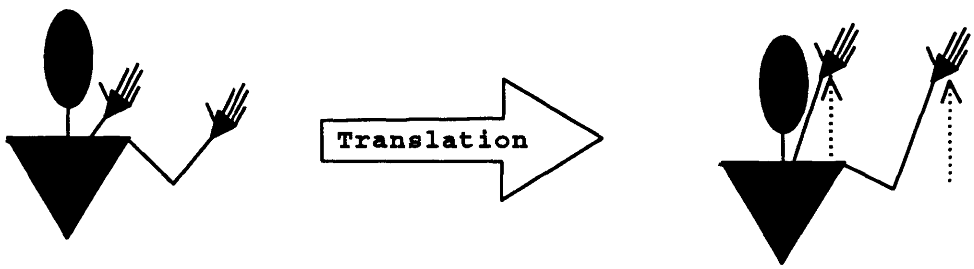

Figure 4-11: Translation Gesture

The scaling gesture consists of indicating the original size of the object with both hands separated, palms facing each other, a certain distance apart, and then indicating the final size of the object in the same manner. See Figure 4.8.

The translation gesture consists of referring to one of the "sides" of the object via holding out both hands, a certain distance apart, palms facing each other. Then, both hands are moved, in parallel, to the new location of the object. The reason which it was decide to have the user specify a side was so that the move could be "scaled," relative to the "size" of that side of the object. Perhaps it could be argued that this is the least "natural" of the gestures chosen. See Figure 4.8. An additional type of translation was developed. It is one of "dynamic" characteristics. The user can specify with one of his hands, a linear path which an object must carry out.

There are three different types of rotation gestures, named wrist-rotations, base-rotations and dual-hand-rotations. Wrist-rotations consist of holding out both hands, with palms facing each other, as always, and rotating the wrists in parallel, in one direction or the other, without moving the location of the hands much at all. See Figure 4.8.

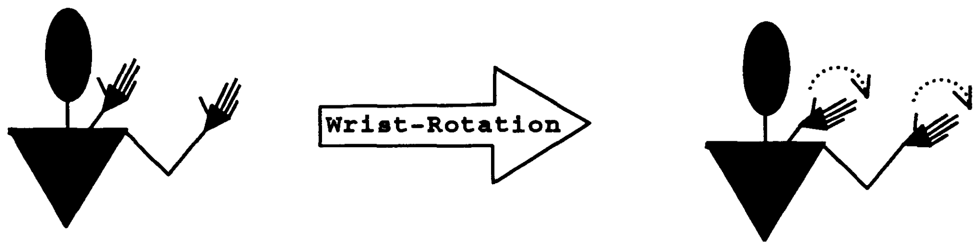

Figure 4-12: Wrist-Rotation Gesture

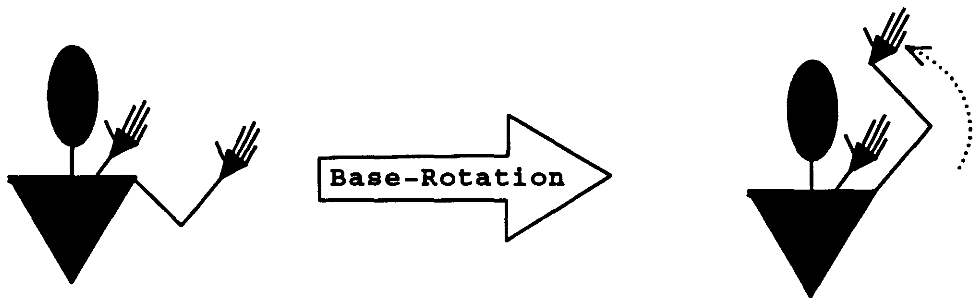

Figure 4-13: Base-Rotation Gesture

Base-rotations consist of holding both hands out, palms facing each other, and then moving only one of the two hands around the other, which stays pretty much in the same place. See Figure 4.8.

Dual-hand-rotations consist of holding both hands out, palms facing each other, and then turning both hands, in parallel, around a circle. The hands typically end up with only one palm facing the other, but they need not. See Figure 4.8.

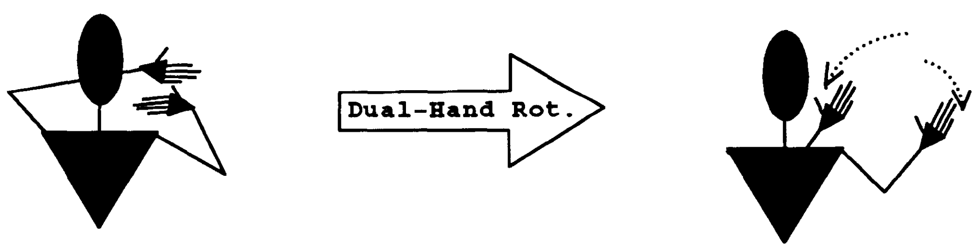

Figure 4-14: Dual Rotation Gesture

4.9 Design Concepts

Users can scale, rotate and translate figures. After the user communicates to the system, via gesticulating, speaking and glancing, it interprets this input. The spirit of the exchange is one of delegation vs. direct manipulation: the user describes how something is to be done to the machine as "agent," rather than perform it themselves in a "hands-on" sense.

4.9.1 "Principal Axes" of an Object

It is standard in computer graphics to refer to different spaces [16]. There is modelling space, the space in which the original object was designed, also referred to as object space. There is world space, the space in which "everything in the world," in the context of the drawing, is represented. There is virtual device-coordinate space, an idealized device coordinate range, and finally there is device-coordinate space, limited by the physical characteristics of the graphics hardware. Any graphical operation performed in one of these spaces must be finally converted into device-coordinate space.

The graphical objects of this system are dealt within object space, as well as world space. Since the moment of their creation onwards, objects keep their "principal axes." "Principal axes" are a determinate number of axes of symmetry of the object. For the current prototypes, they are three mutually perpendicular axes which go through the center of the object [17]; but, the number of axes do not have to be limited to these three.

Rotations and scalings of an object can occur around its "principal axes," regardless of what position and orientation the object has in world space. Translations occur in world space, but they use the object's "principal axes" in an interesting manner: translations are scaled. The user first indicates, which "side" of the object he is referring to with his hands, and the system uses that initial distance between his hands to scale the move accordingly.

[16] Refer to Starbase Techniques Programming Manual [14].
[17] When designing the graphical object attention should be paid to this.

4.9.2 The (Coordinate) "Spaces" of the System

As we have seen, there exist a number of graphics spaces within the system, from virtual spaces to the hardware-dependent device coordinates. Then, there is "real space," the space in which the user gesticulates about. Then, there are approximations to the "real space," the coordinates provided by the left DataGlove, the right DataGlove and the head cube. Each of these spaces has its own origin. Both glove sources, origins, are simply offset by an amount in the y axis. For simplicity's sake, we consider the glove sources almost parallel to the virtual graphics space; the only difference is that the z axis is reversed.

For the head cube, a slightly more rigorous approach is taken, because we must find the intersection of the user's gaze, a vector which "comes out" of the user's eye with the screen. The display screen is "calibrated" in head-cube space, this way a more accurate measurement is made than just assuming the spaces are parallel [18].

Figure 4.9.2 is an attempt to clarify all the mentioned spaces. It should be noted that most of the decisions about the arrangement of these spaces are arbitrary.

4.9.3 Mathematics of "Principal Axes"

All series of graphical transformations, such as every combination of translations, scalings and rotations, can be expressed mathematically as sequences of matrices which are multiplied to produce the current viewing transformation [19]. Quaternions are used for rotations in Mverse. A quaternion is a four parameter quantity representing a vector and a scalar. The quaternion $\vec{q} = (q_1, q_2, q_3, q_0)$ represents an axis in three dimensional space, as well as an additional quantity which can signify the angle of rotation around that axis. The formal definition of quaternion is the algebraic structure which fits the set of all possible rotations.

A quaternion can be used to express the the orientation of an object without the use of trigono-

[18] This approach should also be used with the DataGlove units, for a more robust system.

[19] See Foley and Van Dam for details [11].

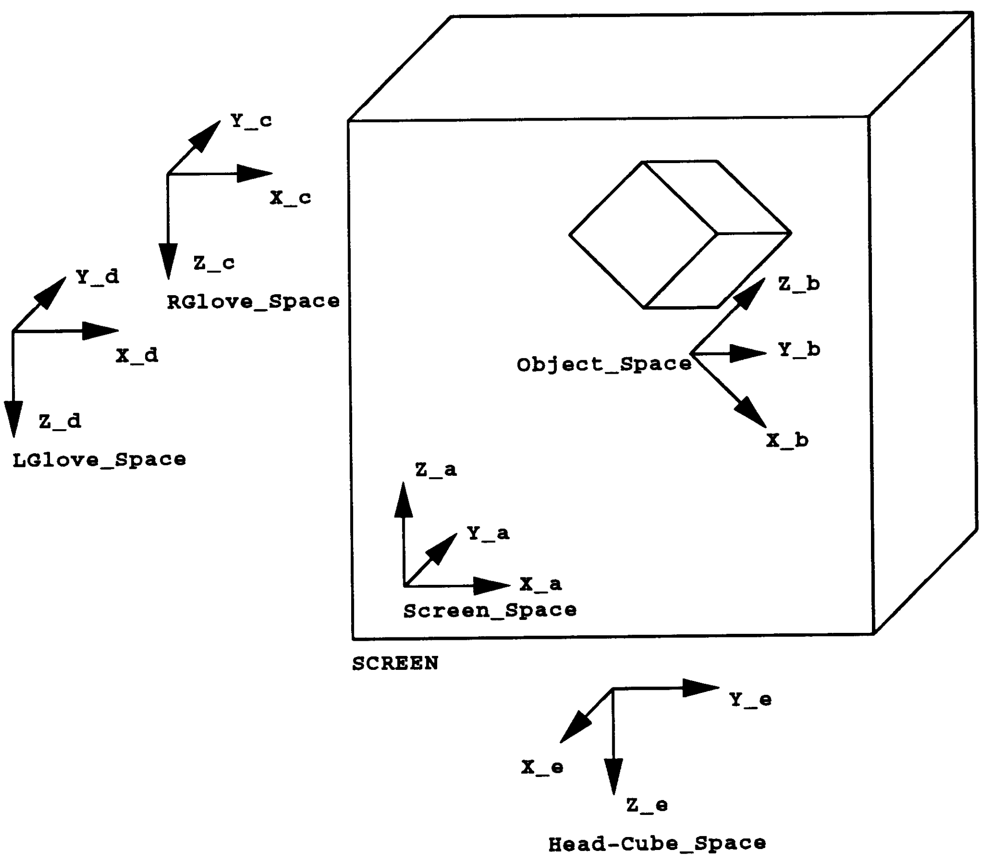

Figure 4-15: All "System Spaces"

metric functions. The orientation of the object is, in a sense, the "location" of the object's original "principal axes" [20] within world space, after the series of graphical transformations in question has been performed. This "orientation" is what is commonly referred to as the attitude matrix of an object.

An attitude matrix has the following form:

$$\begin{pmatrix} a_u & b_u & c_u \\ a_v & b_v & c_v \\ a_w & b_w & c_w \end{pmatrix}$$

And can be viewed as the following:

$$\hat{x'} = a_u\hat{x} + a_v\hat{y} + a_w\hat{z}$$
$$\hat{y'} = b_u\hat{x} + b_v\hat{y} + b_w\hat{z}$$
$$\hat{z'} = c_u\hat{x} + c_v\hat{y} + c_w\hat{z}$$

where $(\hat{x'}, \hat{y'}, \hat{z'})$ are the "principal axes" of the object defined in world space terms.

It is possible to generate the attitude matrix from a quaternion in the following manner [21]: Given a quaternion (e, f, g, h), where (e, f, g) represent the axis, and h the angle of rotation the attitude matrix is as follows:

$$\begin{pmatrix} (h^2 + e^2 - f^2 - g^2) & 2(ef - gh) & 2(eg + fh) \\ 2(gh + ef) & (h^2 - e^2 + f^2 - g^2) & 2(fg - eh) \\ 2(eg - fh) & 2(eh + fg) & (h^2 - e^2 - f^2 + g^2) \end{pmatrix}$$

[20] It is assumed that the "principal axes" of the object are the same as the "world's axes" at the moment of object's definition, or modelling, for simplicity's sake. It was discussed that only three "principal axes" are required, though if more were chosen, then this technique would have to change.

[21] Refer to [27]

4.10 Gesture-Detection Algorithms Details

Notation Details

This section contains pseudocode to describe the gesture-detection algorithms more rigorously than in plain english. The following conventions are followed:

- Functions start with the function name in capital letters, and the parameters, or inputs, in parentheses.
- Lines which begin with $\triangleright$ are comments, the rest of the lines are statements to be executed.
- A '$\leftarrow$' symbol indicates an assignment [22].
- $\vec{A}$ is a vector (A_x, A_y, A_z).
- $\hat{A}$ is a unit vector (A_x, A_y, A_z), i.e., $\sqrt{{A_x}^2 + {A_y}^2 + {A_z}^2}$ equals 1.
- $\vec{AB}$ is a vector $(A_x - B_x, A_y - B_y, A_z - B_z)$.
- Algorithmic constructs are in bold. These are: **do, if, and, then,** and **else**.
- $\vec{u} \times \vec{v}$ represents the cross product of $\vec{u}$ and $\vec{v}$. The cross product of two vectors is another vector perpendicular to both, which follows the "right hand rule." [23] The order of the vectors in the cross product is relevant. The cross product of two parallel vectors is 0.
- $\vec{u} \cdot \vec{v}$ indicates the dot product of vectors $\vec{u}$ and $\vec{v}$. The dot product of two vectors is a scalar, a number. The dot product of two perpendicular vectors is 0.
- DISTANCE($\vec{A}, \vec{B}$) represents the distance between the two points $\vec{A}, \vec{B}$. DISTANCE($\vec{A}, \vec{B}$) = $\sqrt{A_x - {B_x}^2 + A_x - {B_x}^2 + A_x - {B_x}^2}$

[22] The '=' symbol is confusing because one cannot tell for sure if it means assignment or test of equality.
[23] Refer to Strang for a detailed explanation of these principles [32].

4.10.1 Scalings

The first step is when doing a scaling transformation is to calculate the vector between both hands' initial position. The initial distance between both hands and the final distance between them are stored. The vector between both hands is used to find the "principal" axis which the user is referring via doing a "parallelism" test, which is taken care by MAJ_PAR_AXIS(). Then, Mverse is request to scale the object along that axis by the ratio of the final distance to the initial distance.

SCALE($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. $\triangleright$ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $\hat{u} \leftarrow \widehat{AB}$
3. $init\_d \leftarrow$ DISTANCE($\vec{A}, \vec{B}$)
4. $\triangleright \hat{u}$ is a unit vector parallel to $\vec{AB}$, init_d is the initial distance between both hands. $\hat{u}$ is a unitary vector
5. $fin\_d \leftarrow$ DISTANCE($\vec{C}, \vec{D}$)
6. $\hat{v} \leftarrow$ MAJ_PAR_AXIS($\hat{u}$)
7. $\triangleright$ Find out which principal axis the user is referring to
8. Request Mverse to scale object by $\frac{fin\_d}{init\_d}$ along the $\hat{v}$ axis of the object

MAJ_PAR_AXIS() loops through all of the "principal objects" of the object in question and records the modulus of the cross product of the vector which came in as a parameter, and records the value. Upon exit, this function choose the "principal axis" which resulted in a bigger modulus of the discussed cross product, and returns it.

MAJ_PAR_AXIS($\vec{A}$)

1. $\triangleright$ This function returns the "principal axis" of an object which is most parallel to $\vec{A}$
2. $\forall \hat{n} \in \{\hat{u}, \hat{v}, \hat{w}\}$ **do**
3. $\triangleright$ The "principal axes" of the object are $\hat{u}, \hat{v}, \hat{w}$

(a) $\vec{m} \leftarrow \vec{A} \times \hat{n}$
(b) ▷ Take the cross product of $\vec{A}$ and $\hat{n}$
(c) $b_{\hat{n}} \leftarrow \sqrt{{m_x}^2 + {m_y}^2 + {m_z}^2}$

4. $\hat{l} \leftarrow \hat{n}$ in which $b_{\hat{n}}$ has the largest value
5. ▷ This chooses the $\hat{n}$ which is most parallel to $\vec{A}$

4.10.2 Rotations

Rotations are the most complex of the three types of transformations studied. There were three different types of rotations implemented: wrist-rotations, base-rotations and dual-hand-rotations.

WHICH_ROTATION() takes in four points as parameters. They are the initial and final positions of both hands. Its goal is to determine which type of rotation, out of these three, was requested. If both hands hardly move at all, then it is assumed that a wrist-rotation gesture was performed. If both hands moved, then it is assumed that a dual-hand-rotation gesture was performed. Otherwise, it is assumed that a base-rotation gesture was performed. Depending on the gesture WHICH_ROTATION() calls either WRIST_ROTATE(), BASE_ROT(), or DUAL_ROT().

WHICH_ROTATION($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. ▷ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $rhand\_move \leftarrow DISTANCE(\vec{A}, \vec{C})$
3. ▷ quantized movement of right hand
4. $lhand\_move \leftarrow DISTANCE(\vec{B}, \vec{D})$
5. ▷ quantized movement of left hand
6. **if** $rhand\_move$ <threshold **and** $lhand\_move$ <threshold **then** WRIST_ROTATE($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)
7. **else if** $rhand\_move$ >threshold **and** $lhand\_move$ >threshold **then** DUAL_ROT($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)
8. **else** BASE_ROT($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

WRIST_ROTATE() calculates the degrees of wrist rotation, and translates the initial vector between both hands to object space in order to make find out the sign of the rotation. It is explained in the next sub Section why this is done. The degrees of rotation are estimated from the change in yaw of the right hand from the initial to the final position. Mverse is requested to rotate by the degrees calculated the object around the established "principal axis."

WRIST_ROTATE($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. $\triangleright$ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $\hat{u} \leftarrow \widehat{AB}$
3. $\hat{n} \leftarrow$ MAJ_PAR_AX($\hat{u}$)
4. $\alpha \leftarrow$ degrees of wrist rotation
5. Translate $\hat{u}$ to object space to find sign of rotation
6. Make α negative if necessary
7. Request Mverse to Rotate object α degrees around $\hat{n}$

BASE_ROT() and DUAL_ROT() are very similar. DUAL_ROT() calculates the cross product of the initial and final vectors between both hands. The difference is that BASE_ROT() calculates the cross product of the first vector and a vector from the original position of the hand which did not move to the final position of the hand which moved. The resulting cross product vector is fed to MAJ_PAR_AXIS() as input, to discover which "principal axis" the user is referring to. The cross product vector is also transformed into object space, to obtain the correct sign of rotation. The amount of rotation consists of the arc cosine of the dot product of the two vectors which were previously "crossed." Mverse is requested to rotate by the degrees calculated the object around the established "principal axis."

BASE_ROT($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. $\triangleright$ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $\hat{u} \leftarrow \widehat{AB}$

3. $\hat{v} \leftarrow$ either $\widehat{AD}$ or $\widehat{CB}$ depending on which hand moved
4. $\vec{w} \leftarrow \hat{u} \times \hat{v}$
5. ▷ Take the cross-product of the initial and final "hands" vectors
6. $\hat{n} \leftarrow$ MAJ_PAR_AX($\hat{w}$)
7. $\vec{l} \leftarrow \hat{u}$ transformed into object space
8. $\vec{m} \leftarrow \hat{v}$ transformed into object space
9. $\alpha \leftarrow \arccos(\vec{l} \cdot \vec{m})$
10. Request Mverse to Rotate object α degrees around $\hat{n}$

DUAL_ROT() is slightly more flexible than BASE_ROT() because it does not force the assumption that one hand does not move.

DUAL_ROT($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. ▷ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $\hat{u} \leftarrow \widehat{AB}$
3. $\hat{v} \leftarrow \widehat{CD}$
4. $\vec{w} \leftarrow \hat{u} \times \hat{v}$
5. ▷ Take the cross-product of the initial and final "hands" vectors
6. $\hat{n} \leftarrow$ MAJ_PAR_AX($\hat{w}$)
7. $\vec{l} \leftarrow \hat{u}$ transformed into object space
8. $\vec{m} \leftarrow \hat{v}$ transformed into object space
9. $\alpha \leftarrow \arccos(\vec{l} \cdot \vec{m})$
10. Request Mverse to Rotate object α degrees around $\hat{n}$

Correct Sign of Rotations: Transforming into Object Space

In order to obtain the correct sign of rotations, which is typically determined by the arccos() function, it is important to transform the axis of rotation which the user indicates into object space. If the figure has already been rotated it is important that the "principal axis" of the figure and the axis which the user is referring to, are not only matched by a parallelism check, but also have the same direction, otherwise the sign of the rotation would be reversed; this is why a transformation from world space to object space is done.

$(\hat{x}, \hat{y}, \hat{z})$ are the axes of the world space. $(\hat{x'}, \hat{y'}, \hat{z'})$ are the object space axes, which in the current implementations of the system are equivalent the the object's "principal" axes.

The following three equations represent the object space coordinate system in terms of the world space coordinate system.

$$\hat{x'} = a + a_u\hat{x} + a_v\hat{y} + a_w\hat{z}$$
$$\hat{y'} = b + b_u\hat{x} + b_v\hat{y} + b_w\hat{z}$$
$$\hat{z'} = c + c_u\hat{x} + c_v\hat{y} + c_w\hat{z}$$

To go from world space to object space the following matrix product must be performed:

$$\begin{pmatrix} 1 \\ x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a & a_u & a_v & a_w \\ b & b_u & b_v & b_w \\ c & c_u & c_v & c_w \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}$$

4.10.3 Translations

TRANSLATE() takes in four points as input, which consist of the initial and final locations of both hands, as is the case in all of these transformation functions. The initial distance between both hands and the distance of movement of the right hand are recorded. The initial vector between both hands is fed to MAJ_PAR_AX() to see which side of the object the user is referring to, which is reality a "principal axis." Then, Mverse is asked for the "size" of the object along this "axis." Then, Mverse is requested to translate the object in the direction of the vector from the original position of the right hand to the final position of the right hand. The amount of translation along this direction is determined by a double ratio of the distance which the right hand moved to the original distance between both hands to the "size" of the object along the established "principal axis."

TRANSLATE($\vec{A}, \vec{B}, \vec{C}, \vec{D}$)

1. ▷ $\vec{A}$ =Initial location of RGlove, $\vec{B}$ =Initial location of LGlove, $\vec{C}$ =Final location of RGlove, $\vec{D}$ =Final Location of LGlove
2. $\hat{u} \leftarrow \widehat{AB}$
3. $init\_d \leftarrow$ DISTANCE($\vec{A}, \vec{B}$)
4. ▷$\hat{u}$ is a unit vector parallel to $\vec{AB}$, init_d is the initial distance between both hands. $\hat{u}$ is a unitary vector
5. $mv\_d \leftarrow$ DISTANCE($\vec{A}, \vec{C}$)
6. ▷ This is the amount which the right hand moved
7. $g \leftarrow \frac{mv\_d}{init\_d}$
8. $\hat{v} \leftarrow$ MAJ_PAR_AXIS($\hat{u}$)
9. ▷ Find out which principal axis the user is referring to
10. $f \leftarrow$ size of object along $\hat{v}$
11. $g \leftarrow \frac{g}{f}$
12. Request Mverse to translate object in the $\hat{u}$ direction by g units in world space

Paths: "Dynamic" Translations

The other type of translations developed are "dynamic" in nature. They allow a user to specify a linear path which an object must carry out. They are very simple: they consist of finding the direction between the original and final positions of the hand, when the user says something like "That object moves like this," and request mverse to animate the object indefinitely in that direction.

4.11 "Intersection" of Gazes and Gestures: The "Fishing" Story

As a simple demonstration to show that gazes can be "intersected" with gestures, using most of the code developed for this thesis, the "fishing story" was developed. It allows the user to say a phrase such as "The fishing is Lake Tahoe was great" and detects whether the user was looking at his hands or not; if he was looking at his hands, the user was obviously referring to the size of the fish (which the system indicates by displaying a fish on the screen), otherwise any gestures are considered of emphatic nature.

A gross estimation of the intersection of the eye gaze vector with the "gesturing space" is made. See Figure 4.11 to see what the computer displays when it has "decided" that the user was looking at his hands, vs. elsewhere, when he said the sentence "The fishing was great."

This example is a minor milestone in human-computer communication: it is the first time a computer, listening to user speech input, also pays attention to the *eyes* to determine whether the user's hands may or may not be *referential*, and thus convey additional information.

4.12 Sample Interaction Session on Video

Along with this document, a sample interaction session with this system was recorded. The "script" of the video is as follows: First the single-object prototype is demonstrated. The user performs two scalings, three rotations, one of each of the three types: a wrist-rotation, a base-rotation and a dual-hand-rotation. Then, he performs a translation.

The second fragment of the video consists of a demonstration of the dual-object prototype, with eye-tracking. He performs a scaling on one of the objects and a rotation on the other.

Finally, the "fishing story" is shown. See Section 4.11 for details.

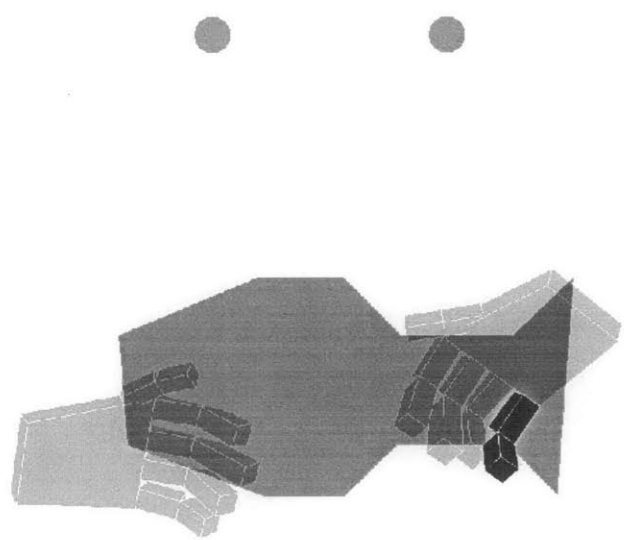

Figure 4-16: "Fishing Story": computer display of the "fish" and user's hands.

Chapter 5

Final Notes and Conclusions

5.1 State-Of-The-Art "Multi-Modal" Equipment

Today's multi-modal detection equipment–speech recognizers, DataGloves, eyetrackers–is not extremely reliable, nor accurate. The hardware in the future ought to be less cumbersome to use. The head-mounted eye-tracker is the most flexible eye-tracking system which has been used at the AHIG, yet it is very tedious to operate. The head-attachment is bulky, and must be fastened tightly to the user's head for correct operation [1]. One cannot simply wear the eye-tracker and use it. A long calibration process must be followed. Eye-tracking will not be practical for everyday use until one can do without calibrations.

For gesture and gaze detection, the ideal hardware would simply consist of a video camera aimed at the user. Image processing on the video signal would be done in real time to "pick out" the position and shape of the user's body as well as his "line of gaze."

An interesting method to pick out viewpoint-independent characteristics of three dimensional objects with neural networks has been studied by Zemel and Hinton [40]. This is an example of

[1] I have had to take many aspirins after long sessions with this device.

the type of image processing necessary to accomplish the development of this futuristic gesture and gaze recognition hardware. Refer to Section C for an introduction to back propagation, the standard method of training neural networks.

5.2 Discussion

The style of manipulating objects implied by this thesis results in relatively "coarse" or approximate positioning, which may be sufficient where the user's aim is that of broad planning or "roughing out" ideas. Where more exact positioning or scaling is needed, approaches include, but are not confined to:

- Agent's knowledge of situation: For example, if we were in the context of rearranging furniture in a room, then the computer "agent," or interpreter, could have "knowledge" about how to locate furniture about, such as having a "sense of proportion," or "being aware" that chairs cannot go upside down.
- "Manipulation space" is not analog (continuous) but is quantized according to different scales of "coarseness."
- Dialogue step: For example, we could say "a little more..., more" to converge on a desired position or scale.

(A fallback is simply to latch onto a mouse!!)

5.3 "Demo"

The system developed for this thesis was "demoed" live for DARPA at the MIT Media Lab, on May 28th, 1992. Out of three demonstration sessions, one worked flawlessly, and the other two had glitches. The reason for these errors, which occurred mainly when demonstrating the rotations was the fact that our DataGloves had been functioning very poorly for the last month, making the "debugging" process very cumbersome. For the day of the "demo" we borrowed the Media Lab's Computer Graphics and Animation's DataGloves.

A video of a sample session is included together with this thesis for reference purposes. See Section 4.12 for a "script" of the video.

5.4 Future Projects

The next logical step to enhance this system would be to develop the capability of grouping and ungrouping objects, to form "higher-level" objects, that one can manipulate. This is not a trivial project, for one would have to decide on a "piecing" scheme: One could allow objects to "go through" each other, or one could implement collision detection methods, or have key "glue dots" [2] on objects which other objects would snap on to (it would be sort of a "virtual Lego" environment).

Another interesting project would be to choose a graphical model of a very complex object, such as a human body, an object which has "too many things" to control with standard interfaces, and adapt the interface from this thesis to be able to control the object. As fellow graduate student Carlton Sparrell has suggested, this would be a very useful tool for computer animators.

5.5 Conclusion

The type of interface implied by this thesis would give scientists, architects, animators, and other professionals who need to visualize geometric constructs, capabilities which would be hard to achieve simply with the use of "traditional" interface tools such as keyboards, mice, knobs, etc. Such an interface would give the user the ability to sit in front of a computer and describe a layout of objects, perform transformations on them, have the ability to group them, all without the need of programming skills.

[2]This was suggested by fellow graduate student David Koons

But this additional "power" need not be for the forementioned professionals; they typically have programming skills, and sometimes even enjoy programming. Non-technologically oriented people would perhaps benefit most from this type of interface.

To my knowledge, this is the first prototype ever developed to integrate *two-handed coverbal* gestures, in a *non direct-manipulation* sense, with speech and gaze. This work's intent was **not** to find a *replacement* for standard interface tools, such as a keyboard or a mouse, but to study the possibility of a system which would be able to interpret "natural" gestures. The goals of this system were to make it appear "natural" to the user, to spare the user from having to learn a new vernacular, and a new operating paradigm.

Bibliography

[1] Richard A. Bolt. *The Human Interface.* Van Nostrand Reinhold, New York, 1984.

[2] Richard A. Bolt. "Put-That-There:" Voice and Gesture at the Graphics Interface. *Computer Graphics*, August 1980 (*SIGGRAPH '80 Proceedings*), Vol. 14, No. 3, 262-270.

[3] Richard A. Bolt and Edward Herranz. Two-handed Gesture with Speech in Multi-Modal Natural Dialog. Internal AHIG document, 1991.

[4] Stewart Brand. *The Media Lab: Inventing the Future at MIT.* Viking Penguin, Inc., New York, 1987.

[5] William Buxton and Brad A. Myers. A Study in Two Handed Input. *CHI '86 Proceedings.*

[6] William Buxton. Haptic Input, Gesture Recognition. Tutorial, CHI '90. Seattle, Washington.

[7] Philip R. Cohen, Mary Dalrymple, Douglas B. Moran, Fernando C. N. Pereira, Joseph W. Sullivan, Robert A. Gargan Jr, Jon L. Schlossberg & Sherman W. Tyler. Synergistic Use of Direct Manipulation and Natural Language. *CHI '89 Proceedings.*

[8] Cosnier, J. (1982). Communications et langages gestuels. In J. Cosnier, J. Coulon, J. Berrendonner, & C. Orecchioni (Eds.), *Les Voies du language: Communications verbales, gestuelles et animales* (pp. 255-303). Paris: Dunod.

[9] Efron, D. (1941/1972). Gesture, race and culture. The Hague: Mouton.

[10] Ekman, P., & Friesen., W. (1972). Hand Movements. *Journal of Communication,* 22, 353-374.

[11] Foley, van Dam, Feiner & Hughes. *Computer Graphics: Principles and Practice.* Second Edition. Addison-Wesley Publishing Company, 1990.

[12] Freedman, N. (1972). The analysis of movement behavior during the clinical interview. In A. R. Siegman & B. Pope (Eds.), *Studies in dyadic communication* (pp. 153-175). Elmsford, NY: Pergamon.

[13] Alexander G. Hauptmann. Speech and Gestures for Graphic Image Manipulation. *CHI '89 Proceedings.*

[14] Hewlett Packard. Starbase Graphics Techniques HP-UX Concepts and Tutorials. Voulumes 1, through 3. HP 900 Series 300/800 Computers. HP, 1988.

[15] Robert J.K. Jacob. Emerging User-Interface Media: Potentials and Challenges, Eye-tracking. Tutorial, SIGGRAPH '89.

[16] Mary Ritchie Key. *Nonverbal Communication: A research Guide & Bibliography.* The Scarecrow Press, Inc., Metuchen, New Jersey, 1977.

[17] Tarun Khanna. *Foundations of Neural Networks.* Addison-Wesley Publishing Company, Inc.

[18] David B. Koons. Resolving Reference in Multi-modal Input. Internal AHIG document, 1991.

[19] B. K. Laurel. Interface as Mimesis, pp. 67-85 in *User Centered System Design*, ed. D. A. Norman and S. W. Draper, Lawrence Erlbaum Associates, Inc., Hillside, NJ (1986).

[20] R. P. Lippman. An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine, April 1987.*

[21] Timothy A. McConnell. Design of a Low Cost, Non-Contact Eye and Head Tracking System. Master's Thesis, MIT Department of Mechanical Engineering.

[22] David McNeill & Elena Levy. Conceptual Representations in Language Activity and Gesture in *Speech, Place and Action*, ed. R. J. Jarvella & W. Klein. John Wiley and Sons Ltd., 1982.

[23] Kouichi Murakami and Hitomi Taguchi. Gesture Recognition using Recurrent Neural Networks. *CHI '89 Proceedings.*

[24] J.G. Neal, C.Y. Thielman, Z. Dobes, S.M. Haller & S.C. Shapiro. Natural Language with Integrated Deictic and Graphic Gestures.1989

[25] Jean-Luc Nespoulous & Andre' Roch Lecours. *Biological Foundations of Gestures: Motor and Semiotic Aspects, Chapter 1*, Current Issues in the Study of Gestures.

[26] Jean-Luc Nespoulous & Andre' Roch Lecours. *Biological Foundations of Gestures: Motor and Semiotic Aspects, Chapter 2*, Gestures: Nature and Function.

[27] Polhemus. A Kaiser Aerospace & Electronics Company. 3Space User's Manual. Colchester, Vermont.

[28] Bernard Rimé & Loris Schiaratura. Gesture and speech, Chapter 7 in *Fundamentals of Nonverbal Behavior*, ed. Robert S. Feldman & Bernard Rimé. Cambridge University Press, Cambridge.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition*, chapter 8, pages 318-362. MIT Press, 1986.

[30] Herbert Schildt. *Artificial Intelligence Using C.* Osborne McGraw-Hill, Berkeley, California, 1987.

[31] Bessie Elizabeth India Starker. A Gaze-directed Graphics World with Synthesized Narration. Master's Thesis, MIT, Dept. of Mechanical Engineering. May, 1989.

[32] Gilbert Strang. *Linear Algebra and its Applications.* Third Edition. Harcourt Brace Jovanovich, Publishers, San Diego, 1988.

[33] David J. Sturman, David Zeltzer, & Steve Pieper. Hands-on Interaction With Virtual Environments, in *Pro. User Interface Software and Technologies '89*, November 13-15, 1989, Williamsburg, VA.

[34] David J. Sturman. Whole-hand Input. Doctoral Dissertation, MIT, Media Arts & Sciences Section. February, 1992.

[35] Kristinn R. Thorisson and David B. Koons. Unconstrained Eye-tracking in Multi-modal Natural Dialogue. Submitted to CHI '92.

[36] VPL Research. DataGlove Model 2, Operation Manual. January, 1989.

[37] David Weimer & S. K. Ganapathy. A synthetic visual environment with hand gesturing and voice input. In *Proceedings of CHI '89 Human Factors in Computing Systems* (Austin, Texas, April 30-May 4, 1989), ACM press, New York, 1990, 235-240.

[38] Patrick H. Winston, Neural Nets (To be included in the next edition of Artificial Intelligence.)

[39] Wundt, W. (1900/1973). *The Language of Gestures.* The Hague: Mouton.

[40] Richard S. Zemel and Geoffrey E. Hinton. View-Point Independent Characteristics of Three Dimensional Objects. Document obtained via 'anonymous ftp' from the University of Toronto. Date of document unknown.

Appendix A

Multiverse Graphics Manager

The mverse [1] program is intended to give high level access to the Starbase resources via the network. Mverse has the ability to load primitives from standard vertex/face files and has the ability to define, load and store classes of objects that can be constructed using primitives and other objects. The program allows you to change attributes of the object including: color in the RGB space, location, orientation, and scaling. It allows animation of objects relative to world coordinate space or relative to any known object's coordinate space, including its own. Animation is coordinated and controlled by a world clock. Mverse also has the ability to pick objects based on references to screen coordinates, both in an explicit and an iterative fashion.

Mverse is set up as a network service to leon. This means that connecting to its reserved socket (15001) on leon (18.85.0.88) will cause the program to be started automatically by inetd. Connection to this socket can be accomplished manually with the telnet command or by programs using the appropriate commands (*socket* and *connect* in the case of C - see /ahi/src/world/sockconn.c for an example).

Please note that in accordance with graphics standards the world coordinate system is right

[1] Christopher R. Wren is the author of this Section. Mverse was developed by Michael Johnson and Christopher R. Wren.

handed, but the camera coordinate system is left handed (x positive to the right, y positive up, and z positive forward—into the screen).

Any command that has "[id]" as an argument will act on the current object if the argument is omitted, or on the object with the object id given.

All commands return a 0 on failure. Success is associated with the appropriate output, or a 1 on commands that don't produce outout. Some commands will also attach error messages, so programs should read the entire line when grabbing results.

Once connection is established, the following commands (with appropriate arguments and a newline) are understood:

A.1 Camera Movement

- **ORBIT** $x\ y\ z$—move the camera in the camera coordinates while leaving the reference point fixed. (ie. attend to the same point in *world* coordinate space while moving.)
- **PAN** $x\ y\ z$—move the camera and the fixation point the same distance and direction in the camera coordinate system. (ie. move camera and attend to the same point in *camera* coordinate space.)
- **SET_EYE** $x\ y\ z$—set the position of the camera to global coordinates (x y z).
- **GET_EYE** $x\ y\ z$—get the position of the camera in global coordinates (x y z).
- **SET_REF** $x\ y\ z$—Set the fixation point of the camera to global coordinates (x y z).
- **GET_REF** $x\ y\ z$—get the fixation point of the camera in global coordinates (x y z).
- **LOOKAT** [id]—Alter the fixation coordinates so that they coincide with center of the current object, or the object with the optional id.
- **AIM** *c deg*—where $c\epsilon\{XYZ\}$ and deg is expressed in degrees. Rotates the camera about the given axis.

- **HOME**—Return camera to default location and fixation in world coordinates. EYE = (0 -50 0) REF = (0 0 0) UP = (0 0 1).

A.2 Database Manipulation

- **LOAD_CLASSES** *file*—given a filename, this command will load the primitives and classes that are listed within. A full pathname is a good idea—but the program will check for the file in /ahi/src/world, and /ahi/demo/mverse for the file if the filename doesn't begin with a slash.
- **SAVE_CLASSES** *path*—new classes are saved in files named: *path/name*.class. The name of the file is then added to the current classfile (the one used in **LOAD_CLASSES**).
- **DEFINE_CLASS** *name*—defines a new class with the given name. All objects currently in the world are made part of the new class. The object will have its center at (0,0,0).
- **RESET**—executes the following commands: HOME, DELETE_ALL, STOP_CLOCK, SET_TIME 0.
- **QUIT**—closes the socket and shutsdown Starbase.

A.3 Object Manipulation

- **SET_CLASS** *classname*—creates a new object at (0,0,0) with orientation (0,0,0) and color 50% grey.
- **GET_CLASS** *[id]*—returns the class of the given object.
- **SET_CURRENT** *id*—sets the current object to the one that has the given id. ID's are assigned in order of creation beginning at one.
- **GET_CURRENT**—Returns the id of the current object - or 0 if there are no objects.
- **DELETE** [id]—delete current object.

- **DELETE_ALL**—deletes all objects. Resets id counter to 1.
- **SET_LOCATION** *x y z [id]*—Places the object at (x,y,z) in world coordinates.
- **SET_COLOR** *r g b [id]*—Changes the color of the object. if the color values are set to -1 then the object will take on any default coloration it might have.
- **SET_ORIENTATION** *qx qy qz qw [id]*

  SET_ORIENTATION *a11 a21 a31 a12 a22 a32 a13 a23 a33 [id]*— Sets the orientation of the object. It is possible to either specify a quaternion or a 3x3 matrix description of the orientation.
- **SET_LABEL** *label [id]*—Sets the objects label to the given string. The string should contain no white space.
- **SET_STATUS** *s [id]*—Sets the status of the object to the given integer.
- **SET_SCALE** *sx sy sz [id]*—Sets the scale of the object in the objects coordinate space.
- **GET_LOCATION, COLOR, ORIENTATION, LABEL, STATUS, SCALE, CLASS**—Returns the requested information about the current object.
- **ROTATE** *qx qy qz qw [id]*—Rotates the object about the given quaternion.
- **TRANS** *x y z [id]*—Translates the object.

A.4 Animation Manipulation

- **SET_TIME** *time*—sets the integer clock to the given time.
- **GET_TIME**—returns the current time on the clock.
- **START_CLOCK**—starts the clock running—allows animation to progress.
- **STOP_CLOCK**—freezes clock at current tick. Freezes all animation.

- **GET_WL**—Prints the current worklist one line at a time with a NULL at the end of the list. Each line contains the fields: id (the object being animated), rid (the object id of the reference frame, or zero for world), dx, dy, dz, qx, qy, qz, dqw, t (time left to the end of the animation).

- **RM_WL** n—will delete the *n*th entry in the worklist. If *n* is zero the worklist will be cleared.

- **ANIMATE** *id start rid n !x y z qx qy qz qw dt ...*—adds animation items to the worklist. New item(s) will cause the object with number *id* to be animated starting at *start* in the coordinate space of object *rid* (or world coords if $rid = 0$.) *N* items will be added to the worklist with start-times that will cause them to happen sequentially. The object will move to the new coordinates and orientation in a linear fashion in the number of frames specified by *dt*. Items on the worklist are started as soon as their start-times are less than the world clock, and continue for the appropriate number of frames regardless of the clock. This means that if items are submitted to the worklist that are tagged with old start-times, they will begin immediately, and possibly concurrently!

  real objects ($id \geq 1$) (x y z) is position and (qx qy qz qw) is a rotation to be carried out.

  camera ($id = -1$) (x y z) is position and (qx qy qz) is point of reference. Both are in reference to the position of the object described by rid.

A.5 Picking Objects From the Screen

- **SET_LOOK** x y r—performs a search for objects in the window centered at the screen coordinates (x,y) with radius of an inscribed circle of r. Returns a count of found objects and the radius. If r is 0.0 then the program performs a search that expands out from the point of interest until at least one object is found. Repeated calls to SET_LOOK with the same x y will continue to expand the search until at least one new object is found or until the entire screen is being searched—for each call to SET_LOOK.

- **GET_LOOK**—returns the next object id in the list of objects that were found to lie in the last search window, or a zero if there are no more objects in the list. The objects are sorted by id, not by "closeness," however, the "found-list" for an expanding search will not contain the objects that were found in the previous call. The search list is cleared before each new search.

Appendix B

Selected Portions of Code

All of this code is in the C programming language.

B.1 geometry.c

```
/* This file contains basic vector operations */

#include "geom.h"

#define square(x) (x)*(x)

/* distance() calculates the distance between vectors pa and
   pb and creates pc, a unitary vector, in the direction from pa
   to pb. */
float distance(pa,pb,pc)
struct threed *pa,*pb,*pc;
{
  double sqrt(),pow();
  float  tempdiff;

  tempdiff  = (float) pow((double)(pa->x - pb->x),2.0);
  tempdiff += (float) pow((double)(pa->y - pb->y),2.0);
  tempdiff += (float) pow((double)(pa->z - pb->z),2.0);
```

```
  tempdiff  = (float) sqrt((double)(tempdiff));
  pc->x=(pa->x - pb->x)/tempdiff;
  pc->y=(pa->y - pb->y)/tempdiff;
  pc->z=(pa->z - pb->z)/tempdiff;
  return(tempdiff);
}

/* modulus() finds the "magnitude" of vector vec */
float modulus(vec)
     struct threed vec;
{
  float  tempdiff;
  double pow(),sqrt();

  tempdiff  = (float) pow((double)vec.x,2.0);
  tempdiff += (float) pow((double)vec.y,2.0);
  tempdiff += (float) pow((double)vec.z,2.0);

  tempdiff  = (float) sqrt((double)(tempdiff));
  return(tempdiff);
}

/* crossproduct() calculates the cross-product of vectors pa and pb
   and stores the result in vector pc */
crossproduct(pa,pb,pc)
struct threed *pa,*pb,*pc;
{
  pc->x=(pa->y*pb->z - pa->z*pb->y);
  pc->y=(pa->z*pb->x - pa->x*pb->z);
  pc->z=(pa->x*pb->y - pa->y*pb->x);
}

/* dotproduct() finds the dot-product of vectors pa and pb and returns
   the result as a float */
float dotproduct(pa,pb)
struct threed *pa,*pb;

{
  float temp=0;
  temp += pa->x*pb->x;
  temp += pa->y*pb->y;
  temp += pa->z*pb->z;
  return(temp);
}
```

B.2 raw.c

```
#include <stdio.h>
#include <math.h>

#include "glove.h"
#include "geom.h"

#define LGLOVE 1
#define RGLOVE 2
#define FALSE 0

extern int rglovehead,lglovehead;
extern GloveRecord **dgright,**dgleft;

int fdleft, fdright;

/* setupGloves() initializes and calibrates a  DataGlove */
setupGloves(which)
int which;
{
  /* Note instead of null, a filename of a calib table can be used */
  InitBox(which, enablePolhemus|enableFlex, NULL);
  if(which == 2)
    printf("right complete\n");
  else
    printf("left complete\n");
/*  get3p(a,b,c,NEW3,which);    */
/* CalibrateGloves( which, NULL);
  printf("got the points\n\n"); */
}

/* GetClosest() obtains a glove record closest in time to the "time"
   input */
int GetClosest(which,time)
int which,time;
{
  int i,oldh;

  if(which==1)
    {
      oldh=lglovehead;
      for(i=0;i<lglovehead;i++){
        if(abs((dgleft[i]->time) - time) <
            abs((dgleft[oldh]->time) - time))
```

```
            oldh=i;
        }
        return(oldh);
    }
  else
    {
        oldh=rglovehead;
        for(i=0;i<rglovehead;i++){
          if(abs((dgright[i]->time) - time) <
              abs((dgright[oldh]->time) - time))
            oldh=i;
        }
        return(oldh);
    }
}
```

B.3 rawcommands.c

```
#include<stdio.h>

#define TRUE 1
#define FALSE 0
#define OBJ_1 1
#define RGLOVE 2
#define LGLOVE 1

/* rec_gest() is used for debugging purposes, it allows to record gesture
   data */
int rec_gest(psent,w)
char **psent;
int w;
{/*
  if(matchstr(psent[0],"start"))
    {
        LogGest = TRUE;
        return(1);
    }
  else if(matchstr(psent[0],"stop"))
    {
        LogGest = FALSE;
```

```
        return(1);
      }
*/
  return(0);
}


/* Transpositions I:  Scaled moves */
/* "Move/locate this/that [object/prism] here/there." */
int recognize_move(fd,psent,wtime,w)
char **psent;
int   *wtime,w,fd;
{
  int matchstr();
  void do_move();

  if(w<5) return(0);

  if(matchstr(psent[0],"move") || matchstr(psent[0],"locate")
     || matchstr(psent[0],"put"))
    if(matchstr(psent[1],"this") || matchstr(psent[1],"that"))
      if(matchstr(psent[2],"object") || matchstr(psent[2],"prism"))
        if(matchstr(psent[3],"this") || matchstr(psent[3],"that"))
          if(matchstr(psent[4],"way") || matchstr(psent[4],"direction"))
          {
            do_move(fd,wtime[1],wtime[0],wtime[4]);
            return(1);
          }
        else return(0);
}


/* Transpositions II:  Paths */
/* "That/this prism/object moves/(is moving) like this/that." */
int recognize_path(fd,psent,wtime,w)
char **psent;
int   *wtime,w,fd;
{
  int matchstr();
  void do_path();

  if(w<5) return(0);

  if(matchstr(psent[0],"this") || matchstr(psent[0],"that"))
    if(matchstr(psent[1],"object") || matchstr(psent[1],"prism"))
```

```
      if(matchstr(psent[2],"moves"))
        if(matchstr(psent[3],"like"))
          if(matchstr(psent[4],"this") || matchstr(psent[4],"that"))
            {
              do_path(fd,wtime[0],wtime[1],wtime[4]);
              return(1);
            }
          else return(0);
}


/* Rotations */
/* "Rotate/turn that/this prism/object (like this/that)/(this way)" */
int recognize_rotation(fd,psent,wtime,w)
char **psent;
int  fd,*wtime,w;
{
  int matchstr();
  void do_rotate();

  if(w<5) return(0);

  if(matchstr(psent[0],"rotate") || matchstr(psent[0],"turn"))
    if(matchstr(psent[1],"this") || matchstr(psent[1],"that"))
      if(matchstr(psent[2],"object") || matchstr(psent[2],"prism"))
        if(matchstr(psent[3],"like"))
          if(matchstr(psent[4],"this") || matchstr(psent[4],"that"))
            {
              do_rotate(fd,wtime[1],wtime[0],wtime[4]);
              return(1);
            }
          else return(0);

}


/* Rescalings */
/* "Scale this object like this/that." */
int recognize_rescaling(fd,psent,wtime,w)
char **psent;
int  *wtime,w,fd;
{
  int matchstr();
  void do_scale();
```

```
  if(w<5) return(0);

  if(matchstr(psent[0],"scale"))
    if(matchstr(psent[1],"this") || matchstr(psent[1],"that"))
      if(matchstr(psent[2],"object"))
        if(matchstr(psent[3],"like"))
          if(matchstr(psent[4],"this")|| matchstr(psent[4],"that"))
            {
              do_scale(fd,wtime[1],wtime[0],wtime[4]);
              return(1);
            }
          else return(0);
}


/* do_move() obtains the correct glove records, selected with respect to
   the speech's timing patterns, and calls translate() */
void do_move(fd,twhich,tbegin,twhere)
     int fd,twhich,tbegin,twhere;
{
  void translate();
  int GetClosest(),irw,ilw,frw,flw;

  printf("---------------------\n");
  printf("do_move \n");
  printf("begin:\n");
  irw=GetClosest(RGLOVE,tbegin);
  ilw=GetClosest(LGLOVE,tbegin);

  printf("where:\n");
  frw=GetClosest(RGLOVE,twhere);
  flw=GetClosest(LGLOVE,twhere);

  translate(fd,OBJ_1,irw,ilw,frw,flw);

  printf("---------------------\n");
}


/* do_path() selects the correct glove records, select with respect to the
   time of key speech "tokens", and calls path() */
void do_path(fd,twhich,tbegin,tpath)
     int fd,twhich,tbegin,tpath;
{
  void path();
```

```
  int GetClosest(),irw,ilw,frw,flw;

  printf("---------------------\n");
  printf("do_path \n");
  printf("begin:\n");
  irw=GetClosest(RGLOVE,tbegin);
  ilw=GetClosest(LGLOVE,tbegin);

  printf("path:\n");
  frw=GetClosest(RGLOVE,tpath);
  flw=GetClosest(LGLOVE,tpath);

  path(fd,OBJ_1,irw,ilw,frw,flw);

  printf("---------------------\n");
}

/* do_rotate() selects the correct glove records, select with respect to the
   time of key speech "tokens", and calls which_rotate() */
void do_rotate(fd,twhich,tbegin,trotmode)
     int fd,twhich,tbegin,trotmode;
{
  int GetClosest(),irw,ilw,frw,flw;
  void base_rot(),wrist_rotate(),dual_rot(),which_rotate();

  printf("---------------------\n");
  printf("do_rotate \n");

  printf("begin:%d\n",tbegin);
  irw=GetClosest(RGLOVE,tbegin);
  ilw=GetClosest(LGLOVE,tbegin);

  printf("scale:%d\n",trotmode);
  frw=GetClosest(RGLOVE,trotmode);
  flw=GetClosest(LGLOVE,trotmode);

  /* for debugging purposes we look at first object */
  which_rotate(fd,OBJ_1,irw,ilw,frw,flw);
  printf("---------------------\n");
}


/* do_scale() selects the correct glove records, select with respect to the
   time of key speech "tokens", and calls scale() */
void do_scale(fd,twhich,tbegin,tscale)
```

```
    int fd,twhich,tbegin,tscale;
{
  int GetClosest(),irw,ilw,frw,flw;
  void scale();

  printf("---------------------\n");
  printf("do_scale \n");

  printf("begin:%d\n",tbegin);
  irw=GetClosest(RGLOVE,tbegin);
  ilw=GetClosest(LGLOVE,tbegin);

  printf("scale:%d\n",tscale);
  frw=GetClosest(RGLOVE,tscale);
  flw=GetClosest(LGLOVE,tscale);

  /* for debugging purposes we look at first object */
  scale(fd,OBJ_1,irw,ilw,frw,flw);
  printf("---------------------\n");
}
```

B.4 rawtransform.c

```
/* Ed H. 5/5/92 */

#include "rotation.h"
#include "geom.h"
#include "glove.h"

#define BUF_SIZE 200

#define SRC_DIFF -8.0 /* Eight inches: Y-difference between both sources */

extern int rglovehead,lglovehead;
extern GloveRecord **dgright,**dgleft;
```

```
/* sign() returns -1 is the input is negative, otherwise it returns 1 */
sign(val)
float val;
{
  if (val<0.0)
    return(-1);
  else
    return(1);
}


/* Translation: Performs a scaled translation.  First the "principal axis"
    which the user is referring to is found, and then the translation is done
    in a "scaled" manner, with respect to the initial distance between the
    user's hands */
void translate(fd,obj_id,initrf,initlf,finrf,finlf)
      int fd,obj_id;
      int initrf,initlf,finrf,finlf;
{
  float init_dist,fin_dist,init_mid_pt,fin_md_pt;
  struct threed ir,il,fr,fl,temp,temp2,cur_pos;
  float distance(),modulus(),d,find_width(),tempf,tempf2;
  void get_cur_pos(),smsg();
  char tmp[BUF_SIZE];

  ir.x = (dgright[initrf]->Pos->x);
  ir.y = (dgright[initrf]->Pos->y);
  ir.z = (dgright[initrf]->Pos->z);

  il.x = (dgleft[initlf]->Pos->x);
  il.y = (dgleft[initlf]->Pos->y) - SRC_DIFF;
  il.z = (dgleft[initlf]->Pos->z);

  fr.x = (dgright[finrf]->Pos->x);
  fr.y = (dgright[finrf]->Pos->y);
  fr.z = (dgright[finrf]->Pos->z);

  fl.x = (dgleft[finlf]->Pos->x);
  fl.y = (dgleft[finlf]->Pos->y) - SRC_DIFF;
  fl.z = (dgleft[finlf]->Pos->z);

  init_dist= distance(&ir,&il,&temp);
  fin_dist = distance(&fr,&fl,&temp);

  /* see which side of the figure user is referring to in order to establish
```

```
    width of object */
  d=find_width(fd,ir,il);

  get_cur_pos(fd,&cur_pos);

  tempf=distance(&fr,&ir,&temp);

  temp2.x = ir.x-il.x;
  temp2.y = ir.y-il.y;
  temp2.z = ir.z-il.z;
  tempf2=modulus(temp2);

  printf("I->(x=%f y=%f z=%f) id=%f F->(x=%f y=%f z=%f) fd=%f\n",
         temp.x,temp.y,temp.z,tempf,temp2.x,temp2.y,temp2.z,tempf2);

  tempf = tempf/(tempf2 * d);

  temp.x =  temp.x * tempf + cur_pos.x;
  temp.y =  temp.y * tempf + cur_pos.y;
  temp.z = -temp.z * tempf + cur_pos.z;

  if(temp.x <=-10.0) temp.x= -10.0;
  if(temp.x >= 10.0) temp.x=  10.0;

  if(temp.y <=-10.0) temp.y= -10.0;
  if(temp.y >= 10.0) temp.y=  10.0;

  if(temp.z <=-10.0) temp.z= -10.0;
  if(temp.z >= 10.0) temp.z=  10.0;

  sprintf(tmp,"SET_LOCATION %f %f %f\n",temp.x,temp.y,temp.z);
  smsg(fd,tmp);
}


/* get_cur_pos() requests Mverse for the location of the current object */
void get_cur_pos(fd,cp)
int fd;
struct threed *cp;
{
  char buf[BUF_SIZE];
  void gmsg(),smsg();
  float x,y,z;

  smsg(fd,"SET_CURRENT 3\n");
```

```
  gmsg(fd,"GET_LOCATION\n",buf);
  sscanf(buf,"%f %f %f",&x,&y,&z);

  cp->x=x;
  cp->y=y;
  cp->z=z;
}


/* Find out which main axis is parallel to ir-il and then
   figure out what d is based on GET_SCALE from mverse */
float find_width(fd,ir,il)
     struct threed ir,il;
     int fd;
{
  int maj_par_ax(),rval;
  struct threed tmp,sc;
  char temp[200];

  tmp.x = ir.x - il.x;
  tmp.y = ir.y - il.y;
  tmp.z = ir.z - il.z;

  rval = maj_par_ax(fd,tmp);

  /* Get current scaling factors */
  gmsg(fd,"GET_SCALE\n",temp);
  sscanf(temp,"%f %f %f",&(sc.x),&(sc.y),&(sc.z));

  switch(rval){
  case 1:
    return(sc.x);
    break;
  case 2:
    return(sc.y);
    break;
  case 3:
    return(sc.z);
    break;
  }
  return(1.0);
}

/* scale() looks at the initial distance between the user's hands, which
```

```
   "principal axis" the user is referring to, and the final distance
   between the users hands and scales accordingly */
void scale(fd,obj_id,initrf,initlf,finrf,finlf)
     int fd,obj_id;
     int initrf,initlf,finrf,finlf;
{
  float idist,fdist,distance(),newx,newy,newz;
  struct threed temp,sc,ir,il,fr,fl;
  void smsg();
  char tmp[BUF_SIZE];
  int maj_par_ax(),rval;


  ir.x = (dgright[initrf]->Pos->x);
  ir.y = (dgright[initrf]->Pos->y);
  ir.z = (dgright[initrf]->Pos->z);

  il.x = (dgleft[initlf]->Pos->x);
  il.y = (dgleft[initlf]->Pos->y) - SRC_DIFF;
  il.z = (dgleft[initlf]->Pos->z);

  fr.x = (dgright[finrf]->Pos->x);
  fr.y = (dgright[finrf]->Pos->y);
  fr.z = (dgright[finrf]->Pos->z);

  fl.x = (dgleft[finlf]->Pos->x);
  fl.y = (dgleft[finlf]->Pos->y) - SRC_DIFF;
  fl.z = (dgleft[finlf]->Pos->z);

  /* Get current scaling factors */
  gmsg(fd,"GET_SCALE\n",tmp);
  sscanf(tmp,"%f %f %f",&(sc.x),&(sc.y),&(sc.z));

  printf("init r (x=%f,y=%f,z=%f)\n",ir.x,ir.y,ir.z);
  printf("init l (x=%f,y=%f,z=%f)\n",il.x,il.y,il.z);
  printf("final r (x=%f,y=%f,z=%f)\n",fr.x,fr.y,fr.z);
  printf("final l (x=%f,y=%f,z=%f)\n",fl.x,fl.y,fl.z);

  idist = distance(&ir,&il,&temp);
  printf("init_dist=%f\n",idist);
  fdist = distance(&fr,&fl,&temp);
  printf("fin_dist=%f\n",fdist);

  rval=maj_par_ax(fd,temp);
```

```
  switch(rval){
  case 1:
    printf("scale in x %f\n",sc.x);
    newx=fdist/idist * sc.x;
    sprintf(tmp,"SET_SCALE %f %f %f\n",newx,sc.y,sc.z);
    smsg(fd,tmp);
    break;
  case 2:
    printf("scale in y %f\n",sc.y);
    newy=fdist/idist * sc.y;
    sprintf(tmp,"SET_SCALE %f %f %f\n",sc.x,newy,sc.z);
    smsg(fd,tmp);
    break;
  case 3:
    printf("scale in z %f\n",sc.z);
    newz=fdist/idist * sc.z;
    sprintf(tmp,"SET_SCALE %f %f %f\n",sc.x,sc.y,newz);
    smsg(fd,tmp);
    break;
  }
}

/* The maj_par_ax() function returns 1 if the axis examined is most
parallel to the object's X-axis, 2 if Y-axis, 3 if Z-axis */
int maj_par_ax(fd,ax)
int fd;
struct threed ax;
{
  struct threed x_ax,y_ax,z_ax,the_ax,temp;
  float xval,yval,zval,modulus();
  int rval;
  int crossproduct();
  char buf[500];

  /*get mverse axes */
  gmsg(fd,"GET_ORIENTATION\n",buf);

  sscanf(buf,"%f %f %f %f %f %f %f %f %f",
         &x_ax.x,&y_ax.x,&z_ax.x,&x_ax.y,&y_ax.y,&z_ax.y,
         &x_ax.z,&y_ax.z,&z_ax.z);
  printf("x_ax = (%f,%f,%f)\n",x_ax.x,x_ax.y,x_ax.z);
  printf("y_ax = (%f,%f,%f)\n",y_ax.x,y_ax.y,y_ax.z);
  printf("z_ax = (%f,%f,%f)\n",z_ax.x,z_ax.y,z_ax.z);

  /* transformed axis */
```

```
  the_ax.x=    ax.x;
  the_ax.y=    ax.y;
  the_ax.z= -  ax.z;

  printf("the_ax = (%f,%f,%f)\n",the_ax.x,the_ax.y,the_ax.z);

  crossproduct(&the_ax,&x_ax,&temp);
  xval = modulus(temp);
  crossproduct(&the_ax,&y_ax,&temp);
  yval = modulus(temp);
  crossproduct(&the_ax,&z_ax,&temp);
  zval = modulus(temp);

  printf("x_val = %f y_val = %f z_val = %f\n",xval,yval,zval);

  if ((xval <= yval) && (xval <= zval)) rval = 1;
  else
    if ((yval <= xval) && (yval <= zval)) rval = 2;
  else
    rval =3;
  return(rval);
}


/* path() peforms an animation of the object based on a vector determined
   by two positions of one of the user's hands, determined by the times of
   elements of speech */
void path(fd,obj_id,initrf,initlf,finrf,finlf)
     int fd,obj_id;
     int initrf,initlf,finrf,finlf;
{
  float distance();
  struct threed temp,way;
  struct threed ir,il,fr,fl,temp2;
  void smsg();
  char tmp[BUF_SIZE];
  float leftdist,rightdist;

  ir.x = (dgright[initrf]->Pos->x);
  ir.y = (dgright[initrf]->Pos->y);
  ir.z = (dgright[initrf]->Pos->z);

  il.x = (dgleft[initlf]->Pos->x);
  il.y = (dgleft[initlf]->Pos->y) - SRC_DIFF;
  il.z = (dgleft[initlf]->Pos->z);
```

```
  fr.x = (dgright[finrf]->Pos->x);
  fr.y = (dgright[finrf]->Pos->y);
  fr.z = (dgright[finrf]->Pos->z);

  fl.x = (dgleft[finlf]->Pos->x);
  fl.y = (dgleft[finlf]->Pos->y) - SRC_DIFF;
  fl.z = (dgleft[finlf]->Pos->z);

  leftdist  = distance(&fl,&il,&temp);
  rightdist = distance(&fr,&ir,&temp2);

  if(leftdist>=rightdist){
    way.x = temp.x;
    way.y = -temp.y;
    way.z = -temp.z;}
  else{
    way.x = temp2.x;
    way.y = -temp2.y;
    way.z = -temp2.z;}

  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 0.0 0.0 0.0 -1\n",
          way.x,way.y,way.z);
  smsg(fd,tmp);

  smsg(fd,"START_CLOCK\n");
}

#define MAGIC_THRESH 5.0  /* This threshold is used to distinguish between
                             the different types of rotations */

/* which_rotate() determines which type of rotation has been requested:
   if both hands hardly move then it's a wrist-rotation, if both of them
   move, then it's a dual-handed rotation, otherwise, it's a base rotation */
void which_rotate(fd,obj_id,initrf,initlf,finrf,finlf)
     int fd,obj_id;
     int initrf,initlf,finrf,finlf;
{
  float right_dist,left_dist,distance();
  struct threed temp,ir,il,fr,fl;
  void wrist_rotate(),base_rot(),dual_rot();

  ir.x = (dgright[initrf]->Pos->x);
  ir.y = (dgright[initrf]->Pos->y);
  ir.z = (dgright[initrf]->Pos->z);
```

```
  il.x = (dgleft[initlf]->Pos->x);
  il.y = (dgleft[initlf]->Pos->y) - SRC_DIFF;
  il.z = (dgleft[initlf]->Pos->z);

  fr.x = (dgright[finrf]->Pos->x);
  fr.y = (dgright[finrf]->Pos->y);
  fr.z = (dgright[finrf]->Pos->z);

  fl.x = (dgleft[finlf]->Pos->x);
  fl.y = (dgleft[finlf]->Pos->y) - SRC_DIFF;
  fl.z = (dgleft[finlf]->Pos->z);

  right_dist = distance(&fr,&ir,&temp);
  left_dist  = distance(&fl,&il,&temp);

  printf("RDIST=%f LDIST=%f\n",right_dist,left_dist);

  if((right_dist<=MAGIC_THRESH) &&(left_dist<=MAGIC_THRESH))
    wrist_rotate(fd,obj_id,initrf,initlf,finrf,finlf);
  else
    if((right_dist>MAGIC_THRESH) &&(left_dist>MAGIC_THRESH))
      dual_rot(fd,obj_id,initrf,initlf,finrf,finlf);
  else
    base_rot(fd,obj_id,initrf,initlf,finrf,finlf);
}


/* wrist_rotate() rotates the object around the specified "principal axis",
   by the amount of "change in yaw" of the right hand cube */
void wrist_rotate(fd,obj_id,initrf,initlf,finrf,finlf)
     int fd,obj_id;
     int initrf,initlf,finrf,finlf;
{
  float idist,fdist,distance(),newx,newy,newz,the_ang;
  struct threed temp,sc,ir,il,fr,fl;
  void smsg(),get_cur_pos();
  char tmp[BUF_SIZE];
  int maj_par_ax(),rval;

  ir.x = (dgright[initrf]->Pos->x);
  ir.y = (dgright[initrf]->Pos->y);
  ir.z = (dgright[initrf]->Pos->z);

  il.x = (dgleft[initlf]->Pos->x);
```

```
il.y = (dgleft[initlf]->Pos->y) - SRC_DIFF;
il.z = (dgleft[initlf]->Pos->z);

fr.x = (dgright[finrf]->Pos->x);
fr.y = (dgright[finrf]->Pos->y);
fr.z = (dgright[finrf]->Pos->z);

fl.x = (dgleft[finlf]->Pos->x);
fl.y = (dgleft[finlf]->Pos->y) - SRC_DIFF;
fl.z = (dgleft[finlf]->Pos->z);

printf("WRIST ROTATE\n");

printf("init r (x=%f,y=%f,z=%f)\n",ir.x,ir.y,ir.z);
printf("init l (x=%f,y=%f,z=%f)\n",il.x,il.y,il.z);
printf("final r (x=%f,y=%f,z=%f)\n",fr.x,fr.y,fr.z);
printf("final l (x=%f,y=%f,z=%f)\n",fl.x,fl.y,fl.z);

fdist = distance(&fr,&fl,&temp);
printf("fin_dist=%f\n",fdist);

printf("temp (x=%f,y=%f,z=%f)\n",temp.x,temp.y,temp.z);

rval=maj_par_ax(fd,temp);

the_ang = ((dgright[finrf]->Att->azim)-(dgright[initrf]->Att->azim))/3.0;

switch(rval){
case 1:
  printf("rotate in x %f\n",the_ang);
  get_cur_pos(fd,&temp);
  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 1.0 0.0 0.0 %f 35\n",temp.x,
          temp.y,temp.z,17.0 * sign(the_ang));
  /* sprintf(tmp,"ROTATE 1.0 0.0 0.0 20.0\n"); */
  smsg(fd,tmp);
  break;
case 2:
  printf("rotate in y %f\n",the_ang);
  get_cur_pos(fd,&temp);
  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 1.0 0.0 %f 35\n",temp.x,
          temp.y,temp.z,17.0 * sign(the_ang));
  /* sprintf(tmp,"ROTATE 0.0 1.0 0.0 20.0\n"); */
  smsg(fd,tmp);
  break;
case 3:
```

```
    printf("rotate in z %f\n",the_ang);
    get_cur_pos(fd,&temp);
    sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 0.0 1.0 %f 35\n",temp.x,
            temp.y,temp.z,17.0 * sign(the_ang));
    /* sprintf(tmp,"ROTATE 0.0 0.0 1.0 20.0\n"); */
    smsg(fd,tmp);
    break;
  }

}
```

B.5 rawwingest.c

```
/* This file has a lots of differences with respect to Carlton's original
   wingest.c
 */

#include <rti/rtiio.h>
#include <fcntl.h>
#include <time.h>
#include <termio.h>
#include <signal.h>
#define R 'R'

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <X11/Xaw/Label.h>
#include <Xw/SText.h>
#include <Xw/RCManager.h>
#include <Xw/PButton.h>
#include <Xw/TextEdit.h>
#include "libXs.h"
#include <stdlib.h>

#include <macros.h>

#undef length

#include <string.h>
#include <stdio.h>
#include <math.h>
```

```
#include "geom.h"
#include "glove.h"

#define FALSE 0
#define TRUE 1
#define SPSEND 10
#define STR_E 2
#define MAX_W 50
#undef  length

#define LGLOVE 1
#define RGLOVE 2

#define SCR_DIFF -8.0 /* Eight inches: Y-difference between both sources */

int speech,dectalk,rclock;
int nonblock_arg = NONBLOCKINGIO;/* uncomment for blocking reads*/
Widget edit;
int spidle = 0;
int letter_times[60];
int ltime_index=0;
void rtn();
void backup();

int gd;  /* mverse socket descriptor */

extern int rglovehead,lglovehead;
extern GloveRecord **dgright,**dgleft;

/* This main() is the principal loop of the entire system */
main(argc, argv)
int argc;
char *argv[];
{
  Widget toplevel;
  Widget gorbag;
  Arg wargs[1];
  void dospeech();
  int dogest();
  void doupdate();
  void periodic(),omain();

  setupGloves(LGLOVE);
  setupGloves(RGLOVE);
```

```
  dectalk = open("/dev/rti0/dectalk",O_WRONLY);
  if(dectalk == -1) perror("Opening dectalk pipe"), exit(-1);
  system("cat /dev/rti0/speech > /dev/null");
  speech = open("/dev/rti0/speech",O_RDONLY | O_NDELAY);
  if(speech == -1) perror("Opening speech pipe"), exit(-1);
  rclock = open("/dev/rti0/clock",O_RDWR);
  if(rclock == -1) perror("Opening clock pipe"), exit(-1);

  /* Start mverse */
  omain(&gd);

  /* create top level widget.  This establishes
     connection with X server. 'toplevel' becomes the
     root of the widget tree */

  toplevel = XtInitialize(argv[0],"Wintest", NULL, 0,
                          &argc, argv);

  XtSetArg(wargs[0], XtNcolumns, 1);

  XtSetArg(wargs[0], XtNeditType, XwtextEdit);
  edit = create_one_widget("edit", toplevel, wargs, 1);

  XtAddInput(speech, XtInputReadMask, dospeech, edit);
  XtAddWorkProc(dogest, edit);
  periodic(NULL, NULL);

  XtAddEventHandler(edit, KeyReleaseMask, FALSE, doupdate, NULL);
  XtRealizeWidget(toplevel);
  XtMainLoop();
}

/* dospeech() reads the speech (words) as it becomes available */
void dospeech(w, fid, id)
Widget w;
int *fid;
XtInputId *id;
{
  char mesg[100];
  int mytime,i;
  XEvent myevent;
  XwTextPosition begpos,endpos;

  begpos = XwTextGetInsertPos(edit);
  if(GetSpeech(speech,mesg,&mytime))
```

```
    {
/*        printf("speech input: %s(%d)\n",mesg,strlen(mesg));*/
      if(XwTextGetInsertPos(edit) == 0)
        XwTextInsert(edit, mesg+1);
      else
        XwTextInsert(edit, mesg);
    }
  endpos = XwTextGetInsertPos(edit);
  for(i=begpos; i< endpos; ++i)
    letter_times[i] = mytime;
  ltime_index = endpos;
  spidle = 0;
}

/* This timer is used to know what the "timestamp" for the speech is if
   it was typed in, instead of spoken */
int UpdateTimeCode()
{
  int nowtime,i;
  XwTextPosition endpos;

  nowtime = GetTime(rclock);
  endpos = XwTextGetInsertPos(edit);
  for(i=ltime_index; i < endpos; ++i)
    letter_times[i] = nowtime;
  ltime_index = endpos;
}

/* This gets the time from the raw RTI clock, typically */
int GetTime(fd)
{
  char buf[8];

  write(fd,"!",1);
  read(fd,buf,7);

  return(atoi(buf));
}

/* This function gets words, or "tokens" */
int GetSpeech(fd, reply, mytime)
int fd;
unsigned char *reply;
int *mytime;
{
```

```
  int dummy=0, nbytes=0;
  int index = 0;
  int done = 0;
  int i;
  unsigned char tcode[9];

/*  printf("Looking for $\n");*/
  if((dummy = read(fd, reply, 1)) < 1)
    {
      perror("Reading leading byte:");
      return(FALSE);
    }

  if(reply[0] != 0x24)    /* 0x24 = '$', 0x21 = '!' */
    return(FALSE);

  read(fd, reply, 1); /*remove other '$'*/

  while(!done)
    {
/*        printf("Looking for word!\n");*/
      if((dummy = read(fd, reply+index, 1)) < 1)
        perror("Reading leading byte:");
      if(reply[index++] == 0x21) /* 0x24 = '$', 0x21 = '!' */
        {
          done = 1;
          reply[index-1] = '\0';
/*          printf(" is this the count? %d\n",reply[0]);*/
          reply[ 0 ] = ' ';
        }
    }

  for(dummy=0; nbytes < 8; )
    {
/*        printf("Looking for time\n");*/
      dummy = read(fd, tcode+nbytes, 8 - nbytes);
      if (dummy > 0) (nbytes += dummy);
      else perror("Reading buffer:");
    }
  for(i=1,*mytime=0;i<8;i++) /*first char read should be a space!*/
    *mytime = *mytime*10 + (int)(tcode[i] - '0');
/*  printf("Word received at %d\n",*mytime); */
  return(TRUE);
}
```

```
/* Update the timer */
void doupdate(w, client_data, event)
Widget w;
caddr_t client_data;
XEvent *event;
{
  UpdateTimeCode();
  spidle = 0;
}


/* this is so that X will pay attention to typing */
void periodic(w, id)
/*Widget w;*/
caddr_t w;
XtIntervalId id;
{
  XwTextPosition pos;
  pos = XwTextGetInsertPos(edit);

  if(pos != 0)
    {
      if( spidle > SPSEND )
        rtn(edit, NULL, NULL, 0);
      else
        spidle++;
    }
  XtAddTimeOut(250, periodic, NULL);
}

/* myparse() is the basic top-level parsing function */
int myparse(buf,blen)
char *buf;
int blen;
{
  char **p_s, **parse();
  int i,w;
  int p_t[20];
  int p_indx = 0;
  int recognize_move(),recognize_path(),rec_gest();
  int recognize_rotation(),recognize_rescaling();

  p_s=parse(buf,&w);

  for(i=0;i<w;i++)
```

```
    {
      p_t[i] = letter_times[p_indx];
      p_indx += 1+strlen(p_s[i]);
      printf("%s (%d) @ %d \n",p_s[i],strlen(p_s[i]),p_t[i]);
    }

  if (!recognize_move(gd,p_s,p_t,w))
    if(!recognize_path(gd,p_s,p_t,w))
      if(!recognize_rotation(gd,p_s,p_t,w))
        if(!recognize_rescaling(gd,p_s,p_t,w))
          if(!rec_gest(p_s,w))
            printf("Sentence not recognized\n");
}


/* The function parse() parses a sentence and counts how many words are in
   it.  It returns an array of words.  A word is similar to a string, with
   '\0' in it.
*/
char **parse(sent,w_num)
char *sent; /* Sentence to be parsed */
int *w_num; /* Keeps track of the number of words in the sentence */
{
        char *word, **wrd_a, **init;
        static  char ter[]=" .?;:!"; /* List of possible termination chars */
        void *malloc();
        char *strtok();
        char *strcpy();

        wrd_a=malloc(sizeof(char *) * MAX_W);
        init=wrd_a;
        word = strtok(sent,ter);
        *w_num=0; /* The 1st word is the 0th one */
        while(word!=NULL)
        {
                /* printf("%s\n",word); */
                *wrd_a=malloc(sizeof(char) * (strlen(word)+STR_E));
                strcpy(*wrd_a,word);
                word = strtok(NULL,ter);
                wrd_a++;
                (*w_num)++;
        }
        return(init);
}
```

```
/* matchstr() returns 1 if both strings are equal, 0 otherwise */
int matchstr(st1, st2)
char *st1, *st2;
{
  int i;

  for(i = 0;; i++)
    {
      if(st1[i] != st2[i])
        {
/*          printf("Not eq: %s <-> %s\n",st1,st2); */
          return(0);
        }
      if(st1[i] == '\0')
        {
/*          printf("Eq: %s <-> %s\n",st1,st2); */
          return(1);
        }
    }
}

/* dogest() reads both DataGloves */
int dogest(w, fid, id)
Widget w;
int *fid;
XtInputId *id;
{
  void smsg(),gmsg();
  float modulus(),retv;
  char temp[200];
  struct threed ir,il;
  extern int rglovehead,lglovehead;
  extern GloveRecord **dgright,**dgleft;

  ReadRecord(LGLOVE, FALSE);
  ReadRecord(RGLOVE, FALSE);

  ir.x = (dgright[rglovehead]->Pos->x);
  ir.y = (dgright[rglovehead]->Pos->y);
  ir.z = (dgright[rglovehead]->Pos->z);

  il.x = (dgleft[lglovehead]->Pos->x);
  il.y = (dgleft[lglovehead]->Pos->y) - SCR_DIFF;
  il.z = (dgleft[lglovehead]->Pos->z);
```

```
  retv = modulus(ir);
  ir.x=10*ir.x/retv;ir.y=10*ir.y/retv;ir.z=10*ir.z/retv;

  retv = modulus(il);
  il.x=10*il.x/retv;il.y=10*il.y/retv;il.z=10*il.z/retv;

  smsg(gd,"SET_CURRENT 1\n");
  sprintf(temp,"SET_LOCATION %f %f %f\n",ir.x,ir.y,-ir.z);
  smsg(gd,temp);

  smsg(gd,"SET_CURRENT 2\n");
  sprintf(temp,"SET_LOCATION %f %f %f\n",il.x,il.y,-il.z);
  smsg(gd,temp);
/*
  printf("L.X=%f L.Y=%f L.Z=%f\n",il.x,il.y,il.z);
  printf("R.X=%f R.Y=%f R.Z=%f\n",ir.x,ir.y,ir.z); */

  smsg(gd,"SET_CURRENT 3\n");

  return(FALSE);
}
```

B.6 sock.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <varargs.h>
#include <string.h>

#include "geom.h"

#define MAX_RD 100

extern int errno;
extern char *sys_errlist[];

/* init_gsock() initializes a socket connection to Mverse on Leon */
init_gsock(s)
int *s;
{
  struct sockaddr saddr1;
  struct sockaddr_in *psaddr_in;
```

```
  int addr1, sta;
  short port1;
  char tmp_str[80];

  char buf[256];
  int rdfds, s_mask,s_bit,n, done = 0, doneeof= 0;

  addr1 = inet_addr("18.85.0.88"); /* Leon's InterNet addr */
  port1= 15001;

  *s = socket(AF_INET, SOCK_STREAM, 0);
  if (*s == -1) { perror("creating socket"); exit(-1); }

  psaddr_in = (struct sockaddr_in *) &saddr1;
  psaddr_in -> sin_family = AF_INET;
  psaddr_in -> sin_addr.s_addr = addr1;
  psaddr_in -> sin_port        = htons(port1);
  sta = connect(*s, &saddr1, sizeof(saddr1));
  if (sta == -1) { perror("connecting"); exit(-1); }

  s_bit  = 1L << *s;              /* Include connected socket. */
  s_mask = s_bit | 1L;            /* (1L is for std. input.) */

  sprintf(buf,"Connected to port\n\n");
  write(0,buf,strlen(buf));

}

/** Utilities to support printing formatted strings to given file */
/** descriptors. These are intended to make life easier when low-level */
/** i/o commands (e.g., write(2)) are being used. In addition, perr() */
/** provides a means quickly printing an error message to a log file */
/** (since no stderr is open for the program). */

pmsg(fd, strng)                 /* 'strng' must be null terminated. */
    int fd;                     /* 'fd' must be open/write-valid. */
    char strng[];
{

  return(write(fd, strng, strlen(strng)));
}

void gmsg(fd,send,buf)
    int fd;
    char send[],buf[];
```

```
{
  void smsg();
  int stat;

  smsg(fd,send);
  printf("gmsg fd=%d\n",fd);
  stat=read(fd,buf,MAX_RD);
  printf("s=%d\n",stat);
  if(stat==-1) perror("Error reading to sock");
  while(stat<0){
    stat=read(fd,buf,MAX_RD);
    if(stat==-1) perror("Error readiing to sock");
    printf("s=%d\n",stat);
  }
}


perr(fd, strng)
     int fd;
     char strng[];
{

  char str1[80];

  sprintf(str1, "%s: %s (%u)\n", strng, sys_errlist[errno], errno);
  return(write(fd, str1, strlen(str1)));
}

void smsg(fd, strng)
int fd;
char strng[];
{
  if(pmsg(fd,strng)==-1) {perror("Writing to socket"); /* exit(-1); */}
}

/* omain() initializes all of the objects in Mverse, the x29 plane, being
   the most important one. the other objects are there for debugging
   purposes */
void omain(gd)
int *gd;
{
  void smsg(),gmsg(),get_cur_pos();
  char temp[200];
  struct threed cp;
```

```
  init_gsock(gd);

  gmsg(*gd,"LOAD_CLASSES classfile\n",temp);
  smsg(*gd,"SET_CLASS cube\n");
  smsg(*gd,"SET_LOCATION -6.0 0.0 0.0\n");
  smsg(*gd,"SET_COLOR 0.3 0.0 0.0\n");
  smsg(*gd,"SET_SCALE 0.25 0.25 0.25\n");

  smsg(*gd,"SET_CLASS cube\n");
  smsg(*gd,"SET_LOCATION 6.0 0.0 0.0\n");
  smsg(*gd,"SET_COLOR 0.0 0.3 0.0\n");      /* Make them black for now */
  smsg(*gd,"SET_SCALE 0.25 0.25 0.25\n");

  smsg(*gd,"SET_CLASS x29\n");
  smsg(*gd,"SET_SCALE 0.25 0.25 0.25\n");
  smsg(*gd,"SET_LOCATION 0.0 0.0 -5.0\n");
  smsg(*gd,"SET_COLOR 0.0 1.0 1.0\n");
  smsg(*gd,"ROTATE 1.0 0.0 0.0 15.0\n");

  smsg(*gd,"START_CLOCK\n");

  smsg(*gd,"SET_CLASS cube\n");
  smsg(*gd,"SET_LOCATION 0.0 20.0 12.0\n");
  smsg(*gd,"SET_COLOR 1.0 0.0 0.0\n");
  smsg(*gd,"SET_SCALE 8.0 0.25 0.25\n");

  smsg(*gd,"SET_CLASS cube\n");
  smsg(*gd,"SET_LOCATION 0.0 20.0 12.0\n");
  smsg(*gd,"SET_COLOR 0.0 0.0 1.0\n");
  smsg(*gd,"SET_SCALE 0.25 0.25 8.0\n");
}
```

B.7 turn.c

```
#include <math.h>
#include "geom.h"
#include "glove.h"

#define R 0
#define L 1
```

```
#define SCR_DIFF -8.0 /* Eight inches: Y-difference between both sources */

extern int rglovehead,lglovehead;
extern GloveRecord **dgright,**dgleft;

void get_cur_pos();

/* find_angle() converts two vectors to object space, and finds the angle
   between them */
float find_angle(fd,vec1,vec2)
int fd;
struct threed vec1,vec2;
{
  float dotproduct(),modulus();
  struct threed x_ax,y_ax,z_ax,tempvec1,tempvec2;
  void gmsg();
  char buf[500];

  /* Switch to Object space first to get sign:
     Get mverse axes */
  gmsg(fd,"GET_ORIENTATION\n",buf);

  sscanf(buf,"%f %f %f %f %f %f %f %f %f",
         &x_ax.x,&y_ax.x,&z_ax.x,&x_ax.y,&y_ax.y,&z_ax.y,
         &x_ax.z,&y_ax.z,&z_ax.z);
  printf("x_ax = (%f,%f,%f)\n",x_ax.x,x_ax.y,x_ax.z);
  printf("y_ax = (%f,%f,%f)\n",y_ax.x,y_ax.y,y_ax.z);
  printf("z_ax = (%f,%f,%f)\n",z_ax.x,z_ax.y,z_ax.z);

  tempvec1.x = dotproduct(vec1,x_ax);
  tempvec1.y = dotproduct(vec1,y_ax);
  tempvec1.z = -dotproduct(vec1,z_ax);

  tempvec2.x = dotproduct(vec2,x_ax);
  tempvec2.y = dotproduct(vec2,y_ax);
  tempvec2.z = -dotproduct(vec2,z_ax);

  return(180.0*(float)acos((double)(dotproduct(tempvec1,tempvec2)/
                           (modulus(tempvec1)*modulus(tempvec2))))/M_PI);
}


/* base_rot() performs a base rotation, by finding out the cross
   product of two vectors from both hands, seeing which "principal axis"
```

```
    they are most parallel to, and then requesting mverse to turn the
    object */
void base_rot(mvfd,objid,initr,initl,finr,finl)
int mvfd,objid;
int initr,initl,finr,finl;
{
  int initial,rval;
  struct threed ir,il,fr,fl,vec1,vec2,temp;
  float distance(),find_angle();
  int crossproduct();
  float the_ang;
  char tmp[500];

  ir.x = (dgright[initr]->Pos->x);
  ir.y = (dgright[initr]->Pos->y);
  ir.z = (dgright[initr]->Pos->z);

  il.x = (dgleft[initl]->Pos->x);
  il.y = (dgleft[initl]->Pos->y) - SCR_DIFF;
  il.z = (dgleft[initl]->Pos->z);

  fr.x = (dgright[finr]->Pos->x);
  fr.y = (dgright[finr]->Pos->y);
  fr.z = (dgright[finr]->Pos->z);

  fl.x = (dgleft[finl]->Pos->x);
  fl.y = (dgleft[finl]->Pos->y) - SCR_DIFF;
  fl.z = (dgleft[finl]->Pos->z);

  printf("BASE ROTATION\n");

  printf("I->(rx=%f ry=%f rz=%f) F->(rx=%f ry=%f rz=%f)\n",
         ir.x,ir.y,ir.z,fr.x,fr.y,fr.z);
  printf("I->(lx=%f ly=%f lz=%f) F->(lx=%f ly=%f lz=%f)\n",
         il.x,il.y,il.z,fl.x,fl.y,fl.z);

  printf("base_rot\n");
  if(distance(&ir,&fr,&temp)<=distance(&il,&fl,&temp))
    initial=R;
  else
    initial=L;

  if(initial==R) printf("initial=R\n");
  else printf("initial=L\n");
```

```
if(initial==R){
  vec1.x = il.x   - ir.x;
  vec1.y = il.y   - ir.y;
  vec1.z = il.z   - ir.z;
  vec2.x = fl.x   - ir.x;
  vec2.y = fl.y   - ir.y;
  vec2.z = fl.z   - ir.z;
}
else{
  vec1.x = ir.x   - il.x;
  vec1.y = ir.y   - il.y;
  vec1.z = ir.z   - il.z;
  vec2.x = fr.x   - il.x;
  vec2.y = fr.y   - il.y;
  vec2.z = fr.z   - il.z;
}
crossproduct(&vec1,&vec2,&temp);

the_ang = find_angle(mvfd,vec1,vec2);
if(initial==L) the_ang = the_ang*-1;

rval=maj_par_ax(mvfd,temp);
switch(rval){
case 1:
  printf("rotate in x %f\n",the_ang);
  get_cur_pos(mvfd,&temp);
  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 1.0 0.0 %f 35\n",temp.x,
          temp.y,temp.z,the_ang);
  /* sprintf(tmp,"ROTATE 1.0 0.0 0.0 20.0\n"); */
  smsg(mvfd,tmp);
  break;
case 2:
  printf("rotate in y %f\n",the_ang);
  get_cur_pos(mvfd,&temp);
  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 1.0 0.0 %f 35\n",temp.x,
          temp.y,temp.z,the_ang);
  /* sprintf(tmp,"ROTATE 0.0 1.0 0.0 20.0\n"); */
  smsg(mvfd,tmp);
  break;
case 3:
  printf("rotate in z %f\n",the_ang);
  get_cur_pos(mvfd,&temp);
  sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 0.0 1.0 %f 35\n",temp.x,
          temp.y,temp.z,the_ang);
  /* sprintf(tmp,"ROTATE 0.0 0.0 1.0 20.0\n"); */
```

```
    smsg(mvfd,tmp);
    break;
  }
}

/* dual_rot() performs a base rotation, by finding out the cross product of
   the initial and final vectors between both hands, seeing which "principal
   axis" they are most parallel to, and then requesting mverse to turn the
   object */
void dual_rot(mvfd,objid,initr,initl,finr,finl)
int mvfd,objid;
int initr,initl,finr,finl;
{
  int initial,rval;
  struct threed ir,il,fr,fl,vec1,vec2,temp;
  float distance(),find_angle();
  int crossproduct();
  float the_ang;
  char tmp[500];

  ir.x = (dgright[initr]->Pos->x);
  ir.y = (dgright[initr]->Pos->y);
  ir.z = (dgright[initr]->Pos->z);

  il.x = (dgleft[initl]->Pos->x);
  il.y = (dgleft[initl]->Pos->y) - SCR_DIFF;
  il.z = (dgleft[initl]->Pos->z);

  fr.x = (dgright[finr]->Pos->x);
  fr.y = (dgright[finr]->Pos->y);
  fr.z = (dgright[finr]->Pos->z);

  fl.x = (dgleft[finl]->Pos->x);
  fl.y = (dgleft[finl]->Pos->y) - SCR_DIFF;
  fl.z = (dgleft[finl]->Pos->z);

  printf("DUAL HAND ROTATION\n");

  printf("I->(rx=%f ry=%f rz=%f) F->(rx=%f ry=%f rz=%f)\n",
         ir.x,ir.y,ir.z,fr.x,fr.y,fr.z);
  printf("I->(lx=%f ly=%f lz=%f) F->(lx=%f ly=%f lz=%f)\n",
         il.x,il.y,il.z,fl.x,fl.y,fl.z);

  printf("dual_rot\n");
```

```
  vec1.x = ir.x - il.x;
  vec1.y = ir.y - il.y;
  vec1.z = ir.z - il.z;
  vec2.x = fr.x - fl.x;
  vec2.y = fr.y - fl.y;
  vec2.z = fr.z - fl.z;

  crossproduct(&vec1,&vec2,&temp);

  the_ang = find_angle(mvfd,vec1,vec2);

  rval=maj_par_ax(mvfd,temp);
  switch(rval){
  case 1:
    printf("rotate in x %f\n",the_ang);
    get_cur_pos(mvfd,&temp);
    sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 1.0 0.0 0.0 %f 35\n",temp.x,
            temp.y,temp.z,the_ang);
    printf("%s\n",tmp);
    /* sprintf(tmp,"ROTATE 1.0 0.0 0.0 20.0\n"); */
    smsg(mvfd,tmp);
    break;
  case 2:
    printf("rotate in y %f\n",the_ang);
    get_cur_pos(mvfd,&temp);
    sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 1.0 0.0 %f 35\n",temp.x,
            temp.y,temp.z,the_ang);
    printf("%s\n",tmp);
    /* sprintf(tmp,"ROTATE 0.0 1.0 0.0 20.0\n"); */
    smsg(mvfd,tmp);
    break;
  case 3:
    printf("rotate in z %f\n",the_ang);
    get_cur_pos(mvfd,&temp);
    sprintf(tmp,"ANIMATE 3 0 0 1 !%f %f %f 0.0 0.0 1.0 %f 35\n",temp.x,
            temp.y,temp.z,the_ang);
    printf("%s\n",tmp);
    /* sprintf(tmp,"ROTATE 0.0 0.0 1.0 20.0\n"); */
    smsg(mvfd,tmp);
    break;
  }
}
```

Appendix C

Back Propagation

The basic neural-network architecture consists of an input layer, a certain number of hidden layers and an output layer. All nodes in the network are fully interconnected from one layer to another. Each node connection has a weight attached to it. The forward step consists of the dot product of the input vector by the weight vector of the first hidden layer. This number is fed as an input to the threshold function, the sigmoid. Then the output of all nodes of the first hidden layer will go through an identical process as the one just described to the next layer, and so on, until the output layer is reached. Then the calculated outputs are compared to the desired outputs, provided with the sample inputs, and the total error is determined. The error is then propagated downwards using the back-propagation method, and finally, corresponding changes are made to all the weights.

At every node the output, which can be considered the dot product of the input vector and the weight vector, is fed as input to a threshold function before it makes it through the next level of the network. Traditionally the sigmoid function has been used, but it is not a requirement.

For a more rigorous treatment of the error-correction procedure, also known as back-propagation, see [38], [29], [17] and [20]. The following is a high-level description of the back-propagation method of training feed-forward networks:

- Initialize Network. All weights are set to random real numbers excluding 0. Set learning rate constant R to an arbitrary value.
- Until performance is satisfactory do:
  - For each example:
    - Forward propagate positive example from input nodes to output node. This is done by calculating the dot product of the inputs with the weights, call that σ, and calculate sigmoid(σ) which is the output and then do the same at the next level until everything is propagated to the top.
    - Back Propagation: compute β for nodes in the output layer using $\beta_z = desired\_output_z - calculated\_output_z$. From now on $calculated\_output = o$.
    - Compute β for all other nodes using: $\beta_j = \sum_k w_{j \to k} o_k (1 - o_k) \beta_k$
    - Compute weight changes using: $\Delta w_{i \to j} = R o_i o_j (1 - o_j) \beta_j$
  - Add up weight changes for each example and change weights.